NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

WEB SERVICES INTEGRATION ON THE FLY

by

Hoe Wai Leong

December 2008

Thesis Advisor:	Don Brutzman
Co-Advisor:	Curtis Blais
Second Reader:	Don McGregor

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Web Services Integration on the Fly			**5. FUNDING NUMBERS**
6. AUTHOR: Hoe Wai Leong			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT

In a net-centric environment, data, tools and people operate in a distributed network. A key research question is whether a software framework can become so usable and intelligent that integration of web services can be done on-the-fly as self-integration. Given data, software agents and supporting software infrastructure, web services integration on the fly means that human coding is not required to integrate web services into a Web Service Architecture. This thesis explores a generic, flexible, scalable, usable and intelligent web services architecture framework that enables sharing and integration of data and tools on the fly. This software framework is a key enabler for systems of systems architecture in a net-centric environment. The envisioned Web Service Architecture Intelligent Framework (WSAIF) is applied to the Modeling, Virtual Environments and Simulation (MOVES) domain. Specifically, the framework is applied to provide the capability to search and retrieve visualization models and their matching behavior models in a collaborative environment.

This thesis elaborates on the design, implementation, deployment and test results of web services for the Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) archive, which is a set of web-based 3D graphics models plus corresponding agent-behaviour models. SAVAGE web services can perform both "find" and "get" operations for models in the archives. SAVAGE web services operations can be composed to form business processes. These business processes can be expressed using modeling techniques such as Web Service Business Process Execution Language (WSBPEL). Future capabilities include semantic activities using Web Ontology Language for Services (OWL-S). The study and comparison of various modeling techniques that enable integration, orchestration and adaptation of composable web services is mentioned. The design and implementation approach matches industry best practices for information architectures. The modeling techniques are essential to and will eventually be used in WSAIF Orchestration and Adaptation components. This thesis further explores how WSAIF software agents, modeling data and supporting software infrastructure can someday enable web services integration on the fly and concludes with recommendations for future work.

14. SUBJECT TERMS Service Oriented Architecture, Web Services Architecture, Semantic Web Services, Software Agents, X3D Graphics, SAVAGE Model Archives			**15. NUMBER OF PAGES** 214
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

This thesis done in cooperation with the MOVES Institute

Approved for public release; distribution is unlimited

WEB SERVICES INTEGRATION ON THE FLY

Hoe Wai Leong
Civilian, DSO National Laboratories, Singapore
B.S., National University of Singapore, 2000

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
December 2008

Author:	Hoe Wai Leong
Approved by:	Don Brutzman Thesis Advisor
	Curtis Blais Co-Advisor
	Don McGregor Second Reader
	Mathias Kolsch Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In a net-centric environment, data, tools and people operate in a distributed network. A key research question is whether a software framework can become so usable and intelligent that integration of web services can be done on-the-fly as self-integration. Given data, software agents and supporting software infrastructure, web services integration on the fly means that human coding is not required to integrate web services into a Web Service Architecture. This thesis explores a generic, flexible, scalable, usable and intelligent web services architecture framework that enables sharing and integration of data and tools on the fly. This software framework is a key enabler for systems of systems architecture in a net-centric environment. The envisioned Web Service Architecture Intelligent Framework (WSAIF) is applied to the Modeling, Virtual Environments and Simulation (MOVES) domain. Specifically, the framework is applied to provide the capability to search and retrieve visualization models and their matching behavior models in a collaborative environment.

This thesis elaborates on the design, implementation, deployment and test results of web services for the Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) archive, which is a set of web-based 3D graphics models plus corresponding agent-behaviour models. SAVAGE web services can perform both “find” and “get” operations for models in the archives. SAVAGE web services operations can be composed to form business processes. These business processes can be expressed using modeling techniques such as Web Service Business Process Execution Language (WSBPEL). Future capabilities include semantic activities using Web Ontology Language for Services (OWL-S). The study and comparison of various modeling techniques that enable integration, orchestration and adaptation of composable web services is mentioned. The design and implementation approach matches industry best practices for information architectures. The modeling techniques are essential to and will eventually be used in WSAIF Orchestration and Adaptation components. This thesis further explores how WSAIF software agents, modeling data and supporting software infrastructure can someday enable web services integration on the fly and concludes with recommendations for future work.

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS

ACL	Agent Communication Language
AI	Artificial Intelligence
API	Application Programming Interface
ASP	Application Service Provider
AT/FP	Anti-Terrorism Force Protection
AUV	Autonomous Underwater Vehicle
CWM	Common Warehouse Metamodel
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
DAML-S	DARPA Agent Markup Language for Services
DES	Discrete Event Simulation
DF	Directory Facilitator
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
DOM	Document Object Model
EA	Enterprise Architecture
EAI	Enterprise Application Integration
ECM	Enterprise Content Management
EDOC	Enterprise Distributed Object Computing
EII	Enterprise Information Integration
ESSI	European Semantic Systems Initiative
ETL	Extract, Transform and Load

EXI	Efficient XML Interchange
FIPA	Foundation of Intelligent Physical Agents
GIS	Geographic Information System
GML	Geography Markup Language
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IE	Internet Explorer
IT	Information Technology
JAAS	Java Authentication and Authorization Service
JAXB	Java Architecture for XML Bindings
JAX-WS	Java API for XML Web Services
JDK	Java Development Kit
JSP	Java Servlet Page
MDA	Model-Driven Architecture
MOF	Meta-Object Facility
MOVES	Modeling, Simulation and Virtual Environment
MVC	Model-View-Controller
OASIS	Organization for the Advancement of Structured Information Standards
ODBC	Open Database Connectivity
OMG	Object Management Group
OOP	Object Oriented Programming
OWL	Web Ontology Language

OWL-DL	Web Ontology Language-Description Logic
OWL-S	Web Ontology Language for Services
PCP	Parameter Constraints Pattern
PDDL	Planning Domain Definition Language
P2P	Peer-to-Peer
QoS	Quality of Service
RAHS	Risk Assessment and Horizon Scanning
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RIF	Rule Interchange Format
SAML	Security Assertion Markup Language
SAVAGE	Scenario Authoring and Visualization for Advanced Graphical Environment
SEFAR	Service Enabled Fusion Architecture Reusable
SFTP	Secure File Transfer Protocol
SMAL	SAVAGE Modeling and Analysis Language
SMP	Signature Mismatch Pattern
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPEM	Software Process Engineering Metamodel
SQL	Structured Query Language
SSL	Secure Socket Layer
SSO	Single Sign-On
SVN	Subversion

SWSA	Semantic Web Services Architecture
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Unique Resource Locator
VRML	Virtual Reality Modeling Language
WFS	Web Feature Service
WFS-T	Web Map Service-Transactional
WMS	Web Map Service
WorkSCo	Workflow with Separation of Concerns
WSA	Web Service Architecture
WSAIF	Web Services Architecture Intelligent Framework
WSBPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Modeling eXecution environment
W3C	World-Wide Web Consortium
XACML	XML Access Control Markup Language
X-KISS	XML Key Information Service Specification
XKMS	XML Key Management Specification
X-KRSS	XML Key Registration Service Specification
XMI	XML Metadata Interchange

XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation
X3D	Extensible 3D

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Special thanks to Associate Professor Don Brutzman for advising and encouraging me to work on such an interesting research topic. I want to thank my co-advisor Mr. Curt Blais and second reader Mr. Don McGregor for giving me professional advice on the subject. Most importantly, I thank my wife Mathitla Tan for her patience and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

Service Oriented Architecture (SOA) is a new trend in software architecture and integration. SOA is a style of architecture, where existing or new functionalities are packaged as services. These services communicate with each other by passing data or by coordinating an activity between one or more services. Such an architecture approach is an enabler for systems of systems architectures where data, tools and people operate in a multi-agency, hierarchical and collaborative environment. Web Services Architecture (WSA) is an example implementation of SOA. With this implementation, business modules are implemented and deployed as web services using Hypertext Transfer Protocol (HTTP) invocations enabled via Simple Object Access Protocol (SOAP) bindings. Web Services Description Language (WSDL) is the World-Wide Web Consortium (W3C) standard to define Web Services in XML. Applying these techniques, the new generation of software integration strategies can be more flexible, and integration solutions can be platform and language independent.

Software integration has been governed by framework standardization to help ensure interoperability. The role of software frameworks is to mediate and coordinate multiple functions. The essential operational requirements for software frameworks are uncompromising reliability, acceptable performance and perhaps some level of design consideration for maintainability and scalability. In other words, robustness is the key for what can be defined as a "good" software framework. The same methodology applies to software frameworks that work with WSA. An example state-of-the-art, web-service based architecture framework is Service Enabled Fusion Architecture Reusable (SEFAR). SEFAR is developed by DSO National Laboratories, Singapore. SEFAR enables orchestration of web services on-the-fly by end users. It also supports sharing of data and tools in a multi-agency environment. The architecture framework is tested and used in the Risk Assessment and Horizon Scanning (RAHS) system deployed in 2007 (Foo *et al.*, 2007). The SEFAR architecture framework as implemented has proven itself to be robust and flexible.

B. PROBLEM DESCRIPTION

Typically, in order for a web service to be integrated with SEFAR, implementation may vary from two weeks to three months of engineering effort. This depends on the type of web services and the mediation and translation logic the engineer needs to “hard code” into the service-associated software agents in the SEFAR orchestration framework. This engineering step needs to be done in order for different web services to be composed together. Thus, a key research question is whether the framework might become so usable and intelligent that integration of web services can be done on-the-fly as self-integration. Given data, software agents and supporting software infrastructure, web services integration on the fly means that coding is not required to integrate web services into a Web Service Architecture.

C. MOTIVATION

WSDL is a XML-based language that provides a model to describle web services. Information about web services’ operations, bindings and addresses is described in WSDL.

There are limitations on the amount of information that WSDL can present to users of web services. As such, using WSDL alone, users do not have clarity on the parameters’ and result’s format and meaning. There is a need for more expressive representations of web services; hence, the motivation for more expressive modeling concepts and implementation such as semantic web services (one example is Web Ontology Language for Services (OWL-S)) and Web Services Business Process Execution Language (WSBPEL).

With advanced modeling techniques such as OWL-S and WSBPEL, the relationship between the services and the relationship between information are made explicit. This increases the effectiveness of equipping new users with better business and technical understanding about the web services and how they fit into the overall business architecture. The benefit to users is that they need less time to understand how each web service works, and how a group of them might works together.

A generic and flexible framework which incorporates the above-mentioned capabilities will likely reduce the cost of maintaining the software framework in a heterogeneous network where scalability and adaptability are the key considerations. The software framework provides some level of automation within a run-time environment. Hence, this approach will enable web services invocation, mediation and monitoring. This reduces memory and programmer's attention required to perform low-level and mundane tasks. The benefit to users is that they can manage the web services more efficiently, and are thus able to better focus on strategic business process considerations and decisions.

Business rules, logic and the constraints of how business components (represented as web services) work with each other can be made explicit and captured in the architecture framework. This information becomes the basis for software agents to perform more intelligent data analysis. This means that technology can become more effective in searching, filtering, interpreting, categorizing and prioritizing right information (e.g. matching web services) for users. Likewise, users are able to perform better as they work with applications facilitated by a more usable and intelligent architecture.

Most importantly, these complex models and business rules of web services can be shared within a net-centric environment. This means that users within the network are well supported by a rich knowledge base.

To the managers, the goals of a usable and intelligent architecture are to increase users' productivity, to reduce cost of operation, to minimize cost of maintenance, and to achieve higher service levels.

D. APPROACH

Using state of the art software engineering practices and SOA-based open standards (e.g. OWL-S), this thesis explores a generic, flexible, scalable, usable and intelligent Web Services Architecture Intelligent Framework (WSAIF). These concepts are applied to the Modeling, Simulation and Virtual Environment (MOVES) domain. The framework contains software agents that automatically interpret and execute semantic web services and orchestration workflows. WSAIF will enable MOVES visualization and behavior models such as Extensible 3D (X3D) Graphics and Discrete Event Simulation (DES) files to be discoverable, sharable, composable and self-integrating using web services in the SOA environment.

E. THESIS ORGANIZATION

Chapter II addresses background work. Chapter III elaborates on the envisioned WSAIF. Chapter IV specifies Savage web services which are the fundamental building blocks of web services architecture for the MOVES domain. Chapter V presents the implementation and test results for Savage web services. Chapter VI elaborates on the various modeling techniques to integrate, orchestrate and adapt a composite web services process. The comparison between the modeling techniques is also discussed. These modeling techniques will be used in WSAIF Orchestration and Adaptation components. It further explains how WSAIF software agents and modeling data can enable web services integration on the fly. The final chapter presents conclusions and recommendations for future work.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

A good understanding of the many related disciplines affecting this work aids in better understanding of the problem. Furthermore, this forms the basis for developing design strategies to realize the envisioned architecture. The section on software architecture elaborates on the influencing architecture qualities, state-of-the-art design strategies, intelligent agents and integration technologies. Amidst the different and incorrect understanding of Service Oriented Architecture (SOA), it is important to re-establish the fundamental principles of SOA. Web Services Architecture (WSA) and web services are realizations of SOA. It is also interesting to understand how semantic web technologies influence SOA design considerations and implementation through semantic web services. Web Services Business Process Execution Language (WSBPEL) is a related technology addressing service orchestration. Web Services Choreography and Web Services Security are important areas related to WSAIF. SAVAGE related technologies such as Extensible 3D Graphics (X3D), SAVAGE Modeling and Analysis Language (SMAL) and Discrete Event Simulation (DES) are also discussed. The chapter ends with description of the tools used in the implementation, testing and deployment activities, namely NetBeans, Subversion (SVN) and Protégé.

B. SOFTWARE ARCHITECTURE

1. Architecture Qualities

In software requirements analysis, defining architecture qualities specific to the operational needs for a software application forms part of non-functional requirements. It is also important to consider tradeoffs while defining architecture qualities. Some of the widely used architecture qualities are reliability, performance, scalability, security and maintainability.

a. Reliability

Reliability of a software application is related to its “uptime” or availability (Gorton, 2006). It is typically measured by mean time between failures, or mean time between recovery of a software application. It is also considered to be the most critical architecture attribute. This is because software applications are expected to be available (or not to fail) during operational hours.

b. Performance

A performance quality requirement defines a metric that states the amount of work an application must perform in a given time (Gorton, 2006). Typically, the performance of a strategic information software application has to be acceptable. It is also important to note that poor performance can deter users from using the software application. In military tactical software applications, performance is a critical architecture attribute in view of the expected high rate of data update.

c. Scalability

Scalability describes how the design of the software infrastructure adapts to increases in usage, transactions and deployment requirements. It is often difficult to validate scalability of a software application because of the large amount of resources required to establish the test scenarios. Thus it is practical to leverage good engineering practices from more mature technological areas. Scalability is an important architecture attribute for software frameworks or middleware deployed to “connect the nodes” in a multiple-agencies environment.

d. Security

Security design considerations include authentication, authorization, confidentiality and integrity. A great deal of work has been performed in this area for Web Services and SOA. Authentication verifies the identity of user. Authorization defines the resources that the authenticated user has access to. Java Authentication and Authorization Service (JAAS) is an example technology solution. Transport-layer encryption (Secure Socket Layer (SSL)) and message-level encryption (XML-Encryption) is typically the solution to ensure data confidentiality. Data integrity can be realized via digital signature. Security is a mandatory requirement for deployed software application. It is also important to note that software application security implementations usually come with architecture performance tradeoffs.

e. Maintainability

Maintainability refers to software application supportability. This measure includes the testability and modifiability of the software application. To put it simply, it measures the ease of making software enhancements and testing, troubleshooting and fixing software issues. Software application maintenance can be preventive (which is preferred) or reactive. Reactive maintenance is necessary when preventive maintenance fails to mitigate a certain technical risk of system failure. Reactive maintenance tasks for mission-critical systems are stressful for Information Technology (IT) engineers. This is because of the required high service level which means low tolerance (ie. short response and recovery) for software issues during operations.

2. Design Patterns and Object Oriented Programming

Object Oriented Programming (OOP) is a programming paradigm. It introduces the concept of classes. Interactions are accomplished by message passing between instantiated objects. Design Patterns by the "Gang of Four" (Gamma *et al.*, 1995) is a collection of reusable Object Oriented design templates widely used in software applications. OOP features such as abstraction and interfaces to software components can be clearly defined within the abstract classes. The implementation of the interfaces is handled by the subclasses. These features add flexibility to the object oriented paradigm. The key quality of design patterns is that they are proven successful "best practices" for software design.

Patterns are classified into three categories: creational, behavioral and structural. Creational patterns deal with creation of objects in a system. They are abstract factory, builder, factory method, prototype and singleton. Behavioral patterns focus on the logic that the objects within the system are managed. Chain of responsibility, command, interpreter, iterator, mediator, memento, observer, state, strategy, visitor and template method are the Behavioral patterns. The popular Model-View-Controller architecture pattern can also be considered as a type of behavioral pattern. Structural patterns describe ways to partition and combine entities of a system. It captures the relationships among entities of a system. Adapter, bridge, composite, decorator, façade, flyweight, half-object plus protocol and proxy are structural patterns.

Although it is a good practice to harness the potential of Design Patterns in software development, over-applied Design Patterns can result in unnecessary performance overhead. This is because of the unnecessary run-time overhead; for example, it takes time to execute a chain of responsibility or visit an entire composite pattern when there is no requirement (either use case and design) to do so. Although design patterns serve as a good practice, they do not guarantee quality source code implementation. In other words, a good developer can produce more efficient and more effective as compared to a less experienced developer. This happens even if both developers are implementing the same design pattern. Furthermore, testing is still required for each use case to verify that the operation is implemented correctly.

3. Model-Driven Architecture (MDA)

MDA is a software design approach used for development of software applications. It provides definition of models, which are a set of guidelines for the structuring of specifications. Models provide abstraction of a software application that allow various stakeholders to reason about the software application from different viewpoints and abstraction levels (Gordon, 2006). MDA is related to various standards, which include Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC), the Software Process Engineering Metamodel (SPEM), and the Common Warehouse Metamodel (CWM). MDA is launched and supported by Object Management Group (OMG).

4. Unified Modeling Language (UML)

UML is a standardized software modeling language used in the field of software engineering. The graphical notation comprises structure, behavior and interaction diagrams. Structure diagrams focus on the elements that are required to be modeled in a software application. Class, component, package, deployment, object and composite structure diagrams are structure diagrams. Behavior diagrams model what happens to the elements within the software application. Activity, state machine and use-case diagrams are behavior diagrams. Interaction diagrams emphasize on the modeling of the flow of control between the elements in the software application. Interaction diagrams include communication, interaction overview, sequence and timing diagrams.

UML activity diagrams can be used to model web services composition (Skogan *et al.*, Web Service Composition in UML, 2004). Extensible Stylesheet Language Transformation (XSLT) is used to transform the UML activity model to a web service composition language such as WSBPEL and Workflow with Separation of Concerns (WorkSCo).

5. Middleware and Interoperability

Middleware refers to a commonly used piece of software that connects two or more software applications so that data can be exchanged between them. It also refers to the software layer that sits between the operating system and the software application it connects to. As such, middleware is typically pertinent to developers and transparent to the user.

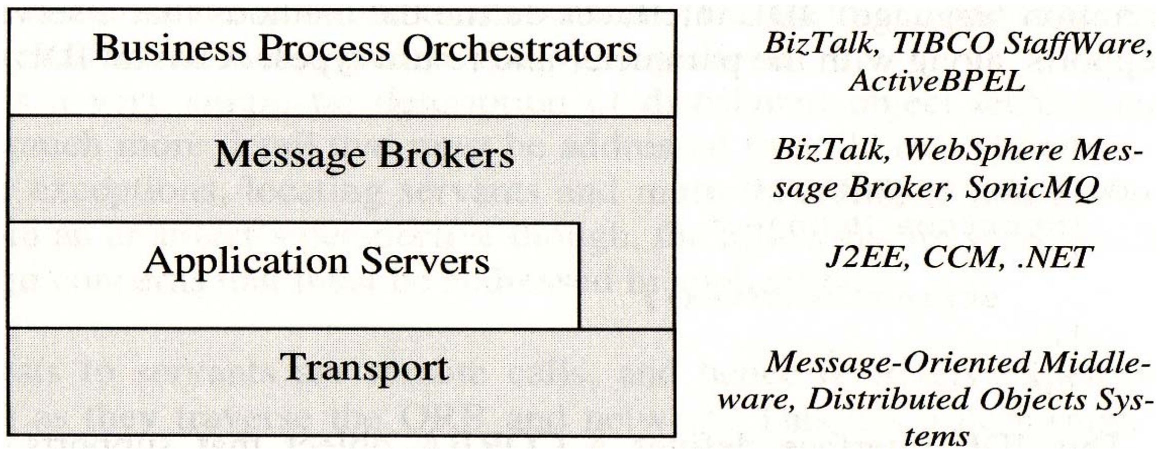

Business Process Orchestrators
BizTalk, TIBCO StaffWare, ActiveBPEL
Message Brokers
BizTalk, WebSphere Message Broker, SonicMQ
Application Servers
J2EE, CCM, .NET
Transport
Message-Oriented Middleware, Distributed Objects Systems

Figure 1. Four classifications of middleware technologies (From Gorton, 2006).

Middleware can be classified into transport, application servers, message brokers and business process orchestrators (Gorton, 2006). The transport layer refers to pipes used for sending requests and for moving data between software components. Examples are distributed object systems and message-oriented middleware. Application servers sit on top of the transport layer and have additional capabilities such as transaction, security and directory services. Some of the examples are BEA Weblogic, JBOSS, Tomcat, IBM Websphere and .NET. Message brokers are software that translates from sender’s formal message protocol to a receiver’s formal message protocol on a network. It leverages the basic capabilities of transport layer and/or application servers. Some examples of message brokers are SonicMQ and WebSphere Message Broker. Business process orchestrators have added capabilities for workflow orchestration for business processes. Some of the examples are BizTalk and ActiveBPEL. However, current implementation of middleware seeks to achieve interoperability given a commonly defined protocol or open standard. Perhaps middleware technology has a greater potential, if systems of systems integration might be accomplished by synergizing and adapting multiple open standards.

6. Software Agents

Software agents are basically software with situated logic that acts on behalf of human users. They exhibit characteristics like responsiveness, pro-activeness and the ability to cooperate with other software agents to achieve multiple objectives. From an architecture perspective, the definition is not sufficiently clear to distinguish between the various application systems (Gorton, 2006). On the other hand, the definition is a good design consideration for categorizing and implementing agent-based functionality and behaviors within an architecture component. In other words, it can be considered as a very specific type of controller in a typical Model-View-Controller (MVC) architecture model.

Rao and Su, includes a survey of automated web service composition methods. The paper mentions that related areas of research basically fall into two realms: namely workflow composition and AI planning. The paper focuses on AI planning, showing that AI planning methods for workflow composition are classified into five categories. They are situation calculus, Planning Domain Definition Language (PDDL), rule-based planning, theorem proving and others. The paper also proposed a general framework for automatic web services composition. The process of automatic service composition consists of five phases, including presentation of single service (ie. advertising atomic services), translation of the languages from external languages (used by service users) to internal languages (used by system), generation of composition process model, evaluation of identified composite services for prioritization, and execution of selected composite service.

C. SERVICE ORIENTED ARCHITECTURE (SOA)

SOA is a style of architecture. Business components are cleanly partitioned and consistently represented as services. The services communicate with each other either by passing data among the services or by coordinating activities among the services. This also establishes a common model for automation logic and business logic. The model applies equally to a task, solution, an enterprise, a community and beyond (Erl, 2005). SOA affords agencies the ability to take advantage of new technologies more easily and respond to end-user demands more quickly and cost-effectively (Matthews, 2008).

The ideal SOA has resources that are decoupled and consistently represented. Resources in an IT architecture context can be data, automation logic, business logic, a task, a solution, an enterprise, a community, and beyond (Erl, 2004). Thus, by adhering to this methodology, coupled with the understanding of fundamental SOA concepts, principles and methodology, open standards such as Web Service Architecture (WSA) and web services offer realizations of the SOA vision. However, according to Thomas Erl, the rise of false SOA has distorted this vision. Many believe that a technical architecture that is service-oriented is simply one that comprises web services. The assumption that the benefits of SOA are attainable solely by investing in web services platform is incorrect. Such perception of a “true path of service-orientation” might further reinforce SOA anti-patterns (bad practices) by further entrenching traditional distributed computing models or, worse, some propriety software solution. Hence, the best way forward is for organizations to have a good understanding and to focus on an ideal IT infrastructure that is transformed by SOA as a style of architecture and work progressively towards aligning systems with the targeted model.

SOA-based applications tend to perform well for strategic applications. Typically, strategic systems sit in a protected environment. With a network infrastructure that includes reliable high capacity bandwidth. On the other hand, there are identified issues when SOA is applied to tactical applications for military use. Establishing and maintaining connectivity between applications and services in a diverse distributed tactical environment can be highly difficult. Furthermore, bandwidth in a wireless tactical environment is limited. Hence, to address these issues, there is a need to rethink data-exchange strategies. Some of the approaches include resolving “small pipe syndrome” through. XML compression, reducing web services calls, batch processing, etc. and also overcoming intermittent connectivity through the use of event driven architecture with robust messaging framework, asynchronous messaging and leverage rich/”smart” client so that functionalities can be retained even if the connection drops (Matthews, 2008).

D. WEB SERVICES ARCHITECTURE (WSA)

WSA is an example variation or realization of SOA using particular standards, as shown in Figure 2.

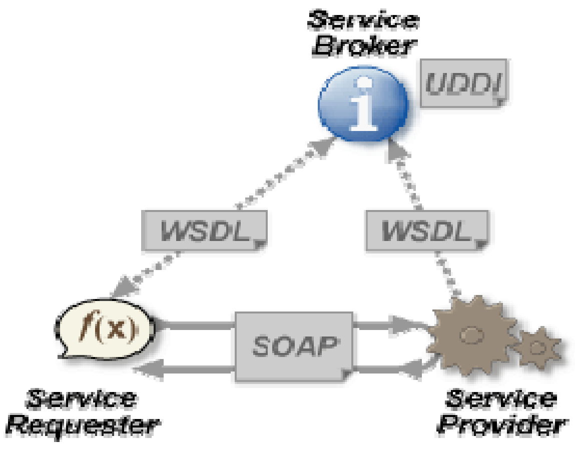
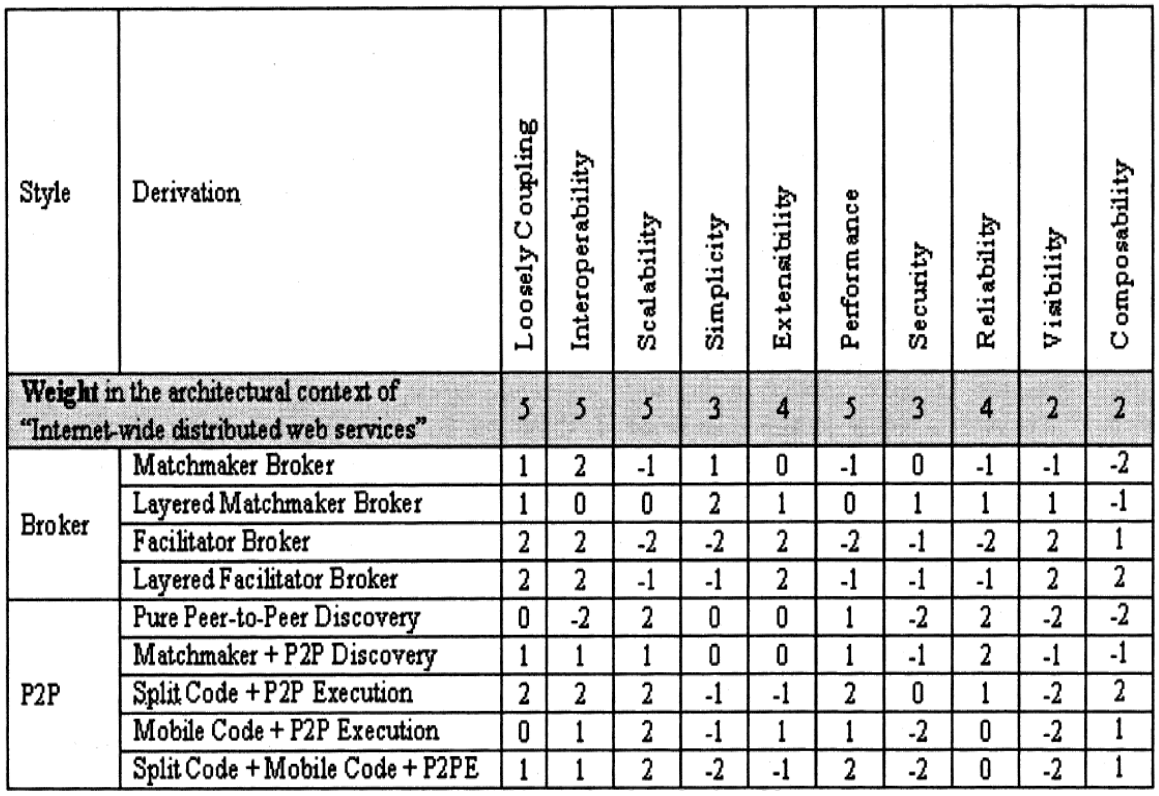


Figure 2. Web Services Architecture consists of service broker, service requester and service provider.

The key components are service provider, service requester and service broker. The service provider refers to the organization deploying the web services. Web Service Description Language (WSDL) descriptions of the web services are registered with the service broker. The service broker is typically realized by open standards such as Universal Description, Discovery and Integration (UDDI). UDDI is a platform-independent, XML-based registry for advertising available web services. The service requester looks up the service broker for information about the web service such as its addresses and endpoints, and then makes http/https invocation of the web service. The protocol which enables such consistent XML-based message bindings is called Simple Object Access Protocol (SOAP).

Wu and Chang have done a comparison between nine WSA styles (Wu and Chang, 2005). They can be broadly categorized into broker-based architecture and peer-to-peer (P2P) architecture. Broker-based architecture includes matchmaker broker, layered matchmaker broker, facilitator broker and layered facilitator broker. P2P architecture includes P2P discovery, matchmaker with P2P discovery, split code with P2P execution and mobile code with P2P execution. Architecture quality properties used for comparison are loose coupling, interoperability, scalability, simplicity, extensibility, performance, security, reliability, visibility and composability. Table 1 povides a summary evaluation of the two styles according to these quality properties.



Style	Derivation	Loosely Coupling	Interoperability	Scalability	Simplicity	Extensibility	Performance	Security	Reliability	Visibility	Composability
Weight in the architectural context of "Internet-wide distributed web services"		5	5	5	3	4	5	3	4	2	2
Broker	Matchmaker Broker	1	2	-1	1	0	-1	0	-1	-1	-2
	Layered Matchmaker Broker	1	0	0	2	1	0	1	1	1	-1
	Facilitator Broker	2	2	-2	-2	2	-2	-1	-2	2	1
	Layered Facilitator Broker	2	2	-1	-1	2	-1	-1	-1	2	2
P2P	Pure Peer-to-Peer Discovery	0	-2	2	0	0	1	-2	2	-2	-2
	Matchmaker + P2P Discovery	1	1	1	0	0	1	-1	2	-1	-1
	Split Code + P2P Execution	2	2	2	-1	-1	2	0	1	-2	2
	Mobile Code + P2P Execution	0	1	2	-1	1	1	-2	0	-2	1
	Split Code + Mobile Code + P2PE	1	1	2	-2	-1	2	-2	0	-2	1

Table 1. Comparison between WSA styles base on architecture quality properties (From Wu and Chang, 2005).

The score is an integer that ranges between -2 to 2 inclusive. The value of this number refers to the degree of which an architecture style exhibits the architecture quality property. Each quality property is assigned a weight from 1 to 5, showing the significance of each architecture quality property with respect to WSA in general.

E. WEB SERVICES

The World Wide Web Consortium (W3C) defines web services as “a software system designed to support interoperable machine to machine interaction over a network.” Basically, web services expose a web application programming interface (API) over the network. WSDL is a XML-based language which provides a model to describe web services. Information about web services’ operations, bindings and addresses is described in WSDL.

It is common to implement business modules, such as data or tools as web services, using a WSA. Web services in this case are the front-end interface or the wrapper to the various business modules. Web services can also be implemented as adapters (data transformers) between two services which do not precisely agree on the syntactic and semantic of application constructs (Harikumar *et al.*, 2005). In this paper, an event driven architecture is explored and components of the architecture include event listener, repository, messaging, pre/post processor, web services (as adapter) and XSLT engine. The XSLT engine is the implemented data transformation logic for the web services.

F. SEMANTIC WEB SERVICES ARCHITECTURE (SWSA)

The Semantic Web Services Architecture (SWSA) committee has identified the scope and potential requirements for a semantic web services architecture (Burstein *et al.*, 2005). The committee has also created a set of architectural and protocol abstractions based on the functional and architectural requirements defined. According to SWSA, phases of semantic web service interaction include candidate service discovery, service engagement (specifically service negotiation and contracts), service process enactment and management, community support services, and quality of service.

G. SWSA AND SOFTWARE AGENTS INTEROPERABILITY

It is interesting to consider the interoperability between software agents and semantic web services. The idea is to introduce middleware “AgentWeb Gateway” to make multi-agent systems standards compatible with existing web services standards without changing their existing specification and implementation (Shafiq *et al.*, 2006).

Solutions that are implemented in AgentWeb Gateway include a service discovery converter which ensures interoperability between Directory Facilitator (DF) and UDDI. A service description converter ensures interoperability between WSDL and DF-Agent A description and communication protocol converter ensures interoperability between Agent Communication Language (ACL) and SOAP. Foundation of Intelligent Physical Agents (FIPA) is an IEEE standards committee and the major specification governing body of Software Agents and Multi Agent Systems. Figure 3 shows the architecture of AgentWeb Gateway middleware that enables compatibility between multi-agent systems standards and existing web services standards.

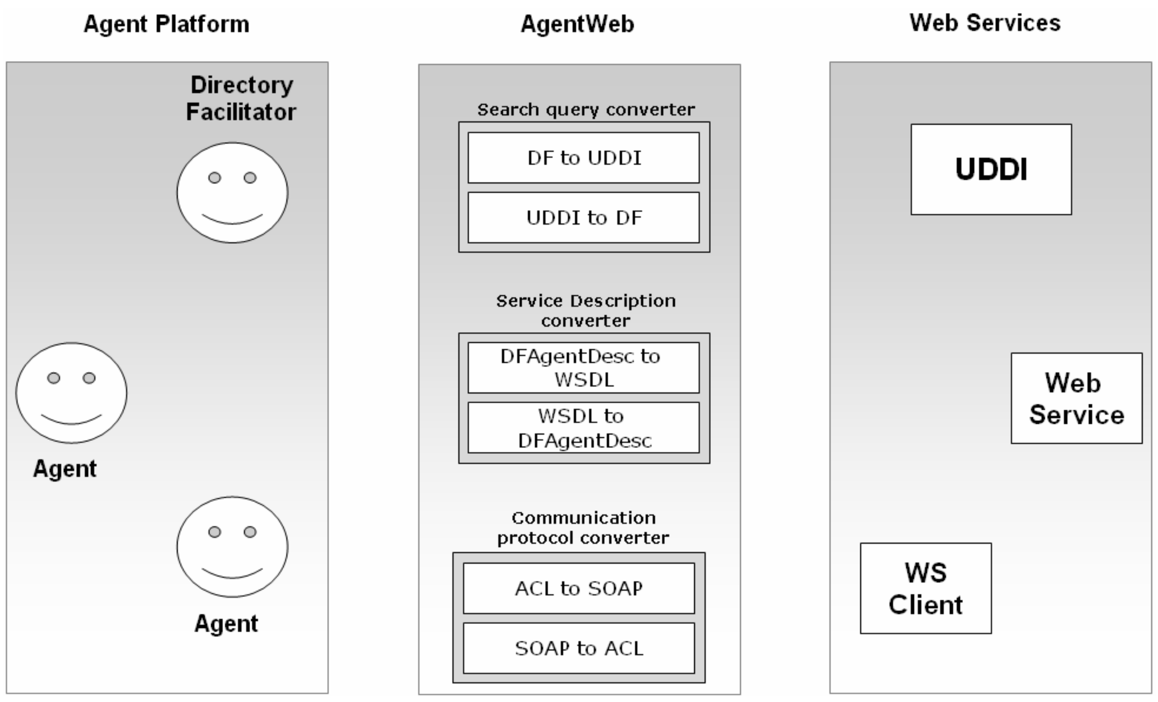


Figure 3. The architecture of AgentWeb Gateway middleware which contains search query converter, service description converter and communication protocol converter. (From Shafiq *et al*, 2006).

H. SEMANTIC WEB TECHNOLOGIES

1. Resource Description Framework (RDF) and RDF Schema (RDFS)

Resource Description Framework (RDF) is a language construct for making assertions about a resource in the form of subject-predicate-object expressions. This model is also called triples. A resource is identified by a unique Uniform Resource Identifier (URI).

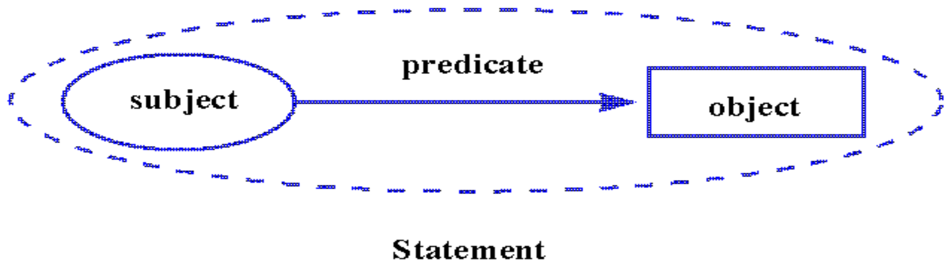


Figure 4. RDF subject-predicate-object model that make assertions about a resource.

Resource Description Framework Schema (RDFS) is an extension of RDF which provides the additional capability of defining classes and class properties. RDFS enables the creation of a vocabulary and has the ability to define class, sub-class, property domain and property range.

RDF and RDFS are both W3C recommended specifications and are the building blocks for defining the Semantic Web (Klyne *et al.* 2004) (Brickley *et al.*, 2004).

2. Web Ontology Language (OWL)

Web Ontology Language (OWL) is an XML based vocabulary that extends RDFS to provide a more comprehensive ontology representation, such as cardinality constraints, quantifiers, etc. Reasoning engines have been developed to check for semantic consistency and help to improve ontology classification. OWL is also a W3C recommended specification. There are three dialects of OWL; namely, OWL-Lite, OWL-DL and OWL-Full. Each dialect has a different level of expressiveness and reasoning capabilities.

a. OWL-Full

OWL-Full is the complete language and was designed to preserve some compatibility with RDFS. Thus, it has no restriction on RDF types used and has the advantage of being useful for modeling a full representation of a domain. However, the trade off is the high complexity of the model. This can result in sophisticated computation that may not complete in finite time.

b. OWL-Description Logic (OWL-DL)

OWL-DL is less expressive than OWL-Full but more expressive than OWL-Lite. OWL-DL has restrictions on the use of some of the description tags. Thus computation formed by a reasoning engine on OWL-DL ontologies can be completed in a finite amount of time (Lacy, 2005). It is also the most commonly used dialect for representing a domain ontology for semantic web applications.

c. OWL-Lite

OWL-Lite is the least expressive compared to OWL-Full and OWL-DL, and is suitable for building ontologies that only require classification hierarchy and simple constraints. In view of its simplicity in expressiveness and constraints capabilities, OWL-Lite provides the most computationally efficient reasoning.

3. Rule Interchange Format (RIF)

The primary goal of Rule Interchange Format (RIF) is to be an effective means of exchanging rules that have the potential to be widely adopted in industry in a way that is consistent with existing W3C technologies and specifications (Paschke and Hirtle, 2008). RIF uses XML as the normative concrete, human-readable syntax.

I. SEMANTIC WEB SERVICES

1. Web Ontology Language for Services (OWL-S)

Web Ontology Language for Services (OWL-S) is an OWL ontology for web services and was originally known as DARPA Agent Markup Language for Services (DAML-S) (Martin *et al.*, 2004). The purpose of this language is to address the limitations of WSDL and UDDI. It is also used to describe rich and flexible metadata required for web services automation such as web services discovery and orchestration. The principal components of an OWL-S description of a service are shown in Figure 5.

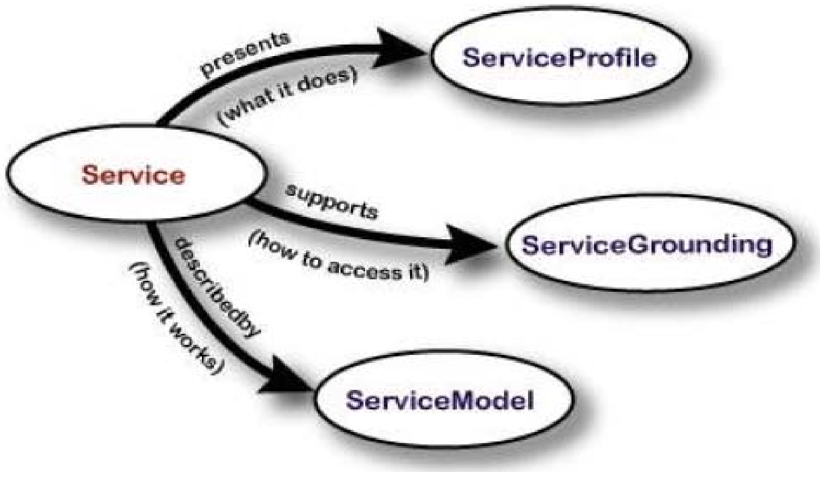


Figure 5. OWL ontology for OWL-S. The class Service *presents* ServiceProfile, is *describedby* ServiceModel, and *supports* ServiceGrounding (From Martin *et al.*, 2004).

ServiceProfile describes "what the service does." Properties used to provide a complete description of serviceProfile include serviceName, intendedPurpose, textDescription, role, provideBy and requestBy. Functional attributes of serviceProfile include geographicRadius, degreeofQuality, serviceParameter, communicationThru, serviceType, serviceCategory, qualityGuarantees and qualityRating.

ServiceGrounding describes "how to access the service" which includes protocol, message format, serialization, transport and addressing.

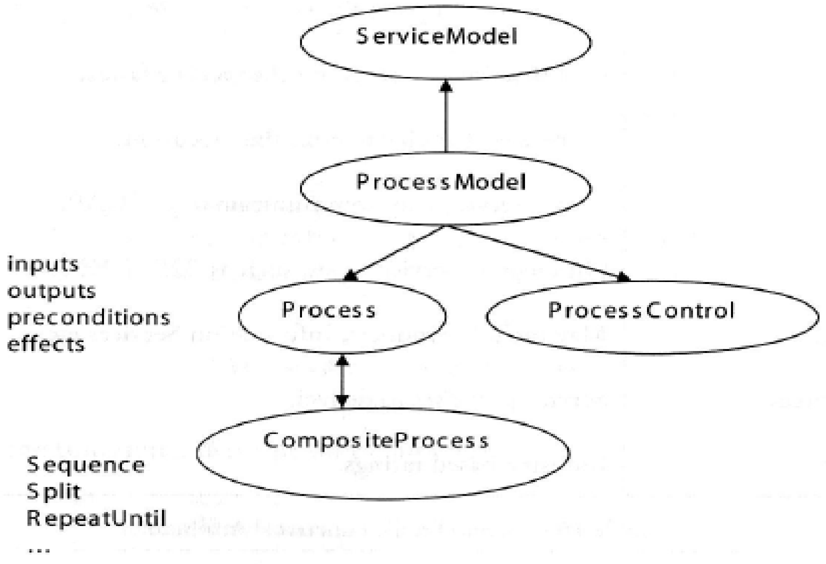


Figure 6. Components of ServiceModel in OWL-S.

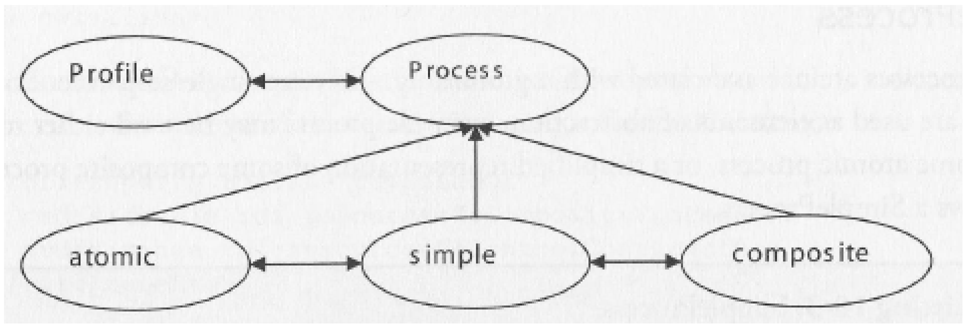


Figure 7. Components of Process Model which is part of ServiceModel.

ServiceModel (Figure 6) describes the process to access a service. The subclass of serviceModel is ProcessModel (Figure 7). ProcessModel has subclasses Process ontology and ProcessControl ontology. There are three types of Process, namely atomic (directly invocable), simple (single-step, but not directly invocable) and composite (made up of other processes). The ProcessControl ontology provides constructs that describe temporal or state dependencies, mapping rules for input state properties to corresponding output state properties, and defining representations for messages about the execution process state. The constructs for ProcessControl ontology are sequence, split, split+join, concurrent, unordered, choice, if-then-else, repeat-until and repeat-while. The ProcessControl ontology is still under development. OWL-S ProcessControl ontology has the potential to be equivalent or better than the service orchestration layer design provided by the Web Service Business Processing Language (WSBPEL). The purpose for OWL-S is to provide sufficiently rich metadata so that software agents are able to read in the data and to automate web service discovery, invocation, workflow orchestration, interoperation and workflow monitoring.

2. Web Service Modeling Ontology (WSMO)

WSMO is a formal ontology used to describe various aspects of semantic web services. The ontology is comprehensive such that it can be exploited by software agents to automate service discovery, composition, execution and interoperation. The WSMO working group is part of the European Semantic Systems Initiative (ESSI) cluster (Bruijin *et al.*, 2005).

There are 4 modeling elements to WSMO: ontologies, web services, goals and mediators. WSMO ontologies consist of non-functional properties, mediators, concept definitions, relation definitions, axioms, and instances (Lara *et al.*, 2004). The purpose of WSMO ontologies is to define the information's formal semantics and allow applications to link machine and human terminologies (Haller *et al.*, 2005).

WSMO goals are basically high-level descriptions of objectives that a service consumer has when he needs to consult web services.

WSMO mediators are one of the most important elements. A mediator's role is to address heterogeneity problems. This refers to syntactic and semantic mismatches between linked elements. Thus, the mediator allows a description of mappings, transformation and reductions between linked elements. There are 4 different types of mediators, namely ggMediator (goal to goal), ooMediator (ontology to ontology), wgMediator (web service to goal) and wwMediator (web service to web service).

WSMO web services provides comprehensive and loose coupling of web services modeling elements. WSMO defines non-functional properties such as performance, quality of service, reliability, security or trust. Definition of WSMO web services also includes the use of WSMO Mediators. Functional capabilities include pre conditions, assumptions, post conditions and effects. WSMO web services also describe details about operation of services, such as error information and compensating services in the event that an error occurs. It defines an orchestration proxy for static and dynamic composition. Message exchange patterns in WSMO web services describe temporal and causal relationships. Finally, WSMO allows description of several groundings for the same web service.

Web Service Modeling Language (WSML) is the formalized modeling language for WSMO. It also provides a rule-based language for the Semantic Web. The WSMO working group includes the WSML working group.

Web Service Modeling eXecution environment (WSMX) is a software architecture that enables creation and execution of semantic web services based on WSMO (Haller *et al.*, 2005). The architecture has the following components (see Figure 8): compiler, matchmaker, data mediator, adaptor, choreography engine, composition and communication manager. The compiler component is responsible for checking the syntactical validity of WSML documents. This component is also used to store parsed information persistently. The matchmaker is used to find suitable services to achieve a goal. The data mediator is the implementation of the ooMediator from the WSMO specification. The adaptor resolves semantic mismatch problems before interoperability between composed services becomes an issue. The choreography engine supports the composition of web services. Choreography of a web service defines the communication pattern which another service/requester has to abide before interacting with the web service. The composition component is used for executing composite/complex composition of web services in order to achieve a goal. The communication manager has two tasks. The component has to handle invocations from requesters. Secondly, it is able to invoke web services, receive and communicate the results back to WSMX. The communication manager is able to handle both synchronous and asynchronous web services calls.

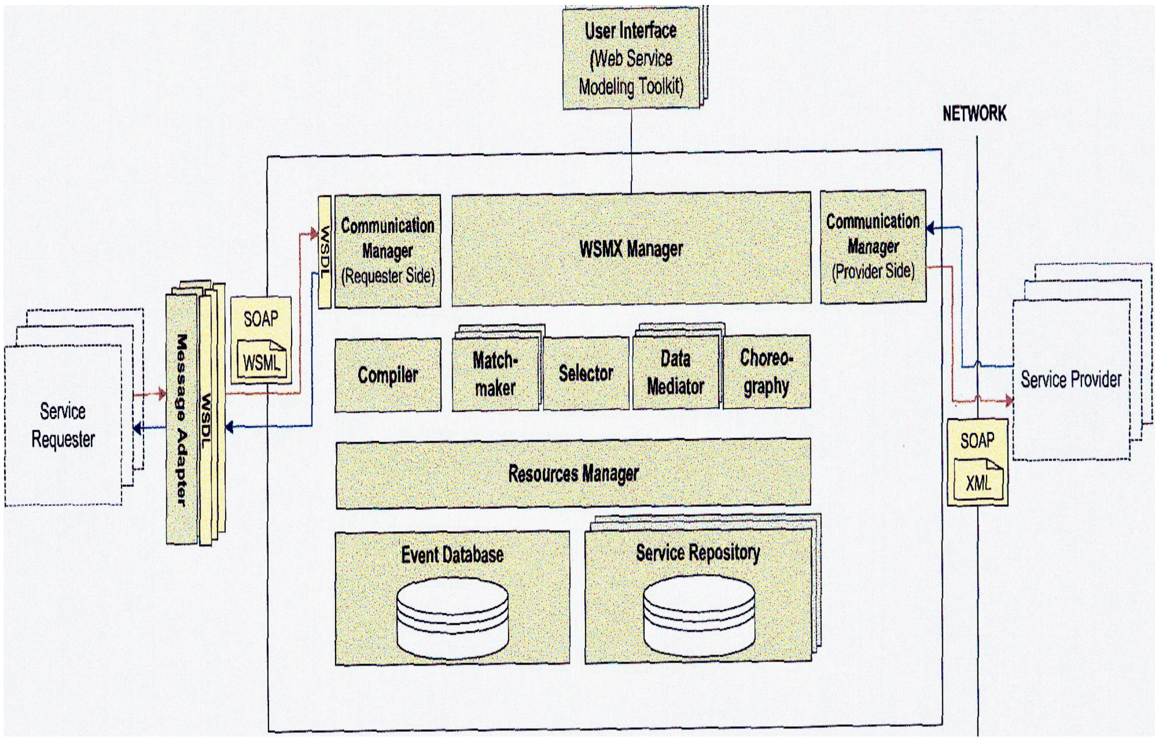


Figure 8. WSMX is a software architecture that enables creation and execution of semantic web services base on WSMO (From Haller *et al.*, 2005).

J. WEB SERVICES BUSINESS PROCESS EXECUTION LANGUAGE (WSBPEL)

WSBPEL focuses on the design of a service orchestration layer by creating business process definitions. It is basically a language to describe business process behavior based on web services. It is formally called BPEL4WS. The release of BPEL4WS 1.0 specification was a joint effort by IBM, Microsoft and BEA. Later, SAP and Siebel Systems joined in for the release of BPEL4WS 1.1. It is currently an Organization for the Advancement of Structured Information Standards (OASIS) open standard. In order to enable capabilities such as automatic web service discovery, orchestration and invocation, WSBPEL works with other open standards such as UDDI and WSDL.

It is important to address the mismatch issue between two services in an orchestrated workflow. Focusing on business level interfaces and protocol, adaptation for replace ability can be achieved (Benatallah *et al.*, 2005). The idea is to make one service compliant to another. Different types of mismatch patterns are identified. At the operational level, there are Signature Mismatch Pattern (SMP) occurs when an operation has the same functionality but differs in operation name, or the number, order or type of input/output parameters. WSBPEL's receive, assign and reply activities are required to resolve such operation-level mismatches. Another operational level mismatch is Parameter Constraints Pattern (PCP) which means differences in value ranges between two operations. WSBPEL's switch, invoke and reply activities are required to resolve this mismatch. A protocol level mismatch refers to extra message, missing message and message split type of mismatches. WSBPEL activities are sufficient to resolve such mismatches.

K. WEB SERVICES CHOREOGRAPHY DESCRIPTION LANGUAGE (WS-CDL)

Web services choreography refers to the sequence of messages between different services to accomplish a flexible composition of services, particularly in an inter-organizational business process. Choreography can be global or local. The global model of choreography specifies the message exchanges from an overall point of view and the local model defines the message interactions from the perspective of one party (Mendling and Hafner, 2004). WS-CDL is an XML-based language that describes peer-to-peer collaboration protocols based on web services. The common and complementary observable behavior to achieve a common business goal is defined from the global viewpoint. The behavior here refers to the order of message exchanges.

There is a distinction between choreography and orchestration. Orchestration is defined by one party and refers to the execution part of the inter-organizational business process. There are multiple parties involved in choreography.

L. WEB SERVICES SECURITY (WS-SECURITY)

Web services identification, authentication and authorization are enabled by the WS-Security standards. Security tokens assert claims and can be used to assert the binding between authentication keys and security identities. An authority can vouch for or endorse the claims in a security token by using its key to sign or encrypt (it is recommended to use a keyed encryption) the security token thereby enabling the authentication of the claims in the token. Thus, authentication proves the identity and authorization states the extent to which the authentication applies. Specifications that enable these mechanisms include SOAP Message Security published by OASIS. In WS-Security, Security Assertion Markup Language (SAML), .NET Passport and XML Access Control Markup Language (XACML) are the three primary extensions that support the implementation of single sign-on (SSO).

For message confidentiality, transport level encryption (which can be handled by the conventional secure socket layer (SSL)) and message-level encryption (specified in XML-Encryption standards recommended by W3C) need to be considered. XML encryption can be applied to parts of a SOAP header and/or SOAP message body. Block-encryption algorithms that can be used in the framework include AES, 3DES and RSA. SOAP message integrity can be ensured via XML-Signature, which is a W3C recommended standard. The specification allows arbitrary cryptographic signature and message authentication algorithms, symmetric and asymmetric authentication schemes, and key agreement methods.

For web services security policies, the WS-Policy framework provides the means to attach properties such as rules, behaviors, requirements and preferences to web services. Such individual properties are represented by policy assertions. Assertions are communicated non-negotiable and preferred policies. WS-Policy can be incorporated within majority WS-* extensions. WS-Policies incorporate assertions with respect to WS-Security, WS-Trust and WS-SecureConversation. The result is the governing standards for web services WS-SecurityPolicy.

For key management for SOAP messages (messages in XML format), the XML Key Management specification (XKMS) is the governing standard. The purpose of XKMS is to specify protocols for distributing and registering public keys. XKMS 2.0 comprises of 2 parts: namely XML Key Information Service Specification (X-KISS) and XML Key Registration Service Specification (X-KRSS).

XML document-centric security is an interesting approach to XML security, which is essential to WS security. The enabler is a security architecture that is expected to provide confidentiality, integrity, and authentication commensurate with the nature of the generated document, maintaining the information objects at an appropriate level of security and acceptable level of risk, as discussed in (Williams, 2008). That thesis aims to investigate the possibility of standardizing XML-based secure document and message dissemination among multinational coalition partners or a multi-agency Homeland Defence task force.

M. VISUALIZATION AND BEHAVIOR MODELING

1. Extensible 3D (X3D) Graphics

Extensible 3D (X3D) Graphics is the ISO Standard for representing 3D computer graphics (see Figure 9). It is a royalty-free open-standard file format and run-time specification. It also has the capability to encode scene graphs in XML. X3D is the successor to the Virtual Reality Modeling Language (VRML) and encompasses extensions to VRML such as Humanoid Animation, GeoVRML, etc. Other features of X3D include its ability to integrate with web services and distributed networks. The standard is componentized, extensible, embedded application ready, real-time and well specified (Web3D, 2008). In order words, 3D graphics on web is made portable using X3D. X3D is the selected visualization technology used in Autonomous Underwater Vehicle (AUV) Workbench and Anti-Terrorism Force Protection (AT/FP) projects with Naval Postgraduate School. There are various editors for X3D, including X3D-Edit which is adapted from earlier designs by the Naval Postgraduate School. The tool has advanced features such as collaboration and version control support which better equip developers in a team-based development environment (Brutzman and Daly, 2007).

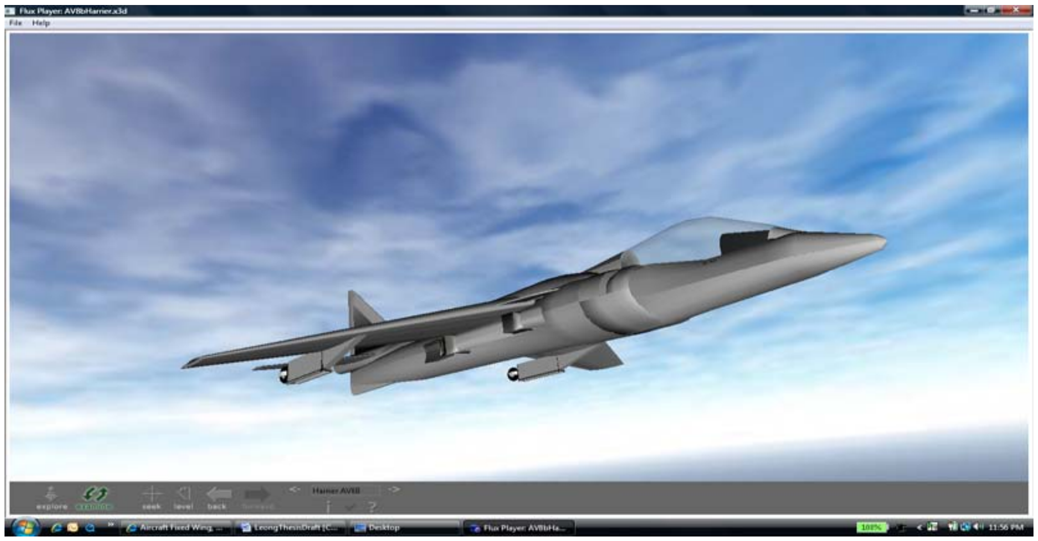
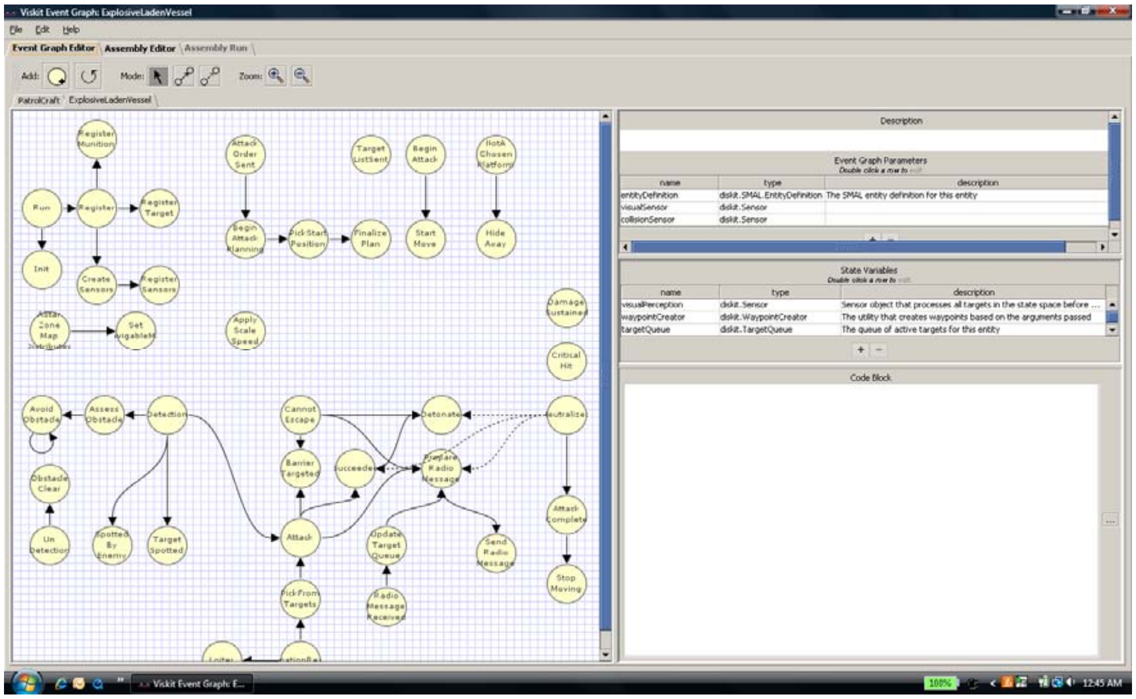


Figure 9. Example of X3D visualization model.

2. Discrete Event Simulation (DES)

Discrete event simulation (DES) is triggered via scheduled events which can have arbitrary durations between them. When handling each event, necessary calculations are made, entity states are updated and new events are added into the schedule. The simulation time is advanced directly to the next event on the event queue. Hence, DES has the advantage of streamlining computation as compared to time-step based simulation. Time-step based simulation incurs overhead at each time step even if there is no new simulation event to perform. DES generally results in better performance from an architectural perspective for many classes of problems.

Event graphs are used to design DES models (Schruben, 1983). To facilitate development of DES models, the Naval Postgraduate School developed SIMKIT and VISKIT. SIMKIT is a Java API for creating DES models. VISKIT (Figure 10) is a visual development environment for SIMKIT and has java code generation capabilities. DES models created using VISKIT are also represented in XML. With these capabilities, VISKIT adds value by increasing the productivity of developers who create and integrate DES models.



Event Graph Parameters
Double click a row to edit

name	type	description
entityDefinition	diskit.SMAL.EntityDefinition	The SMAL entity definition for this entity
visualSensor	diskit.Sensor	
collisionSensor	diskit.Sensor	

State Variables
Double click a row to edit

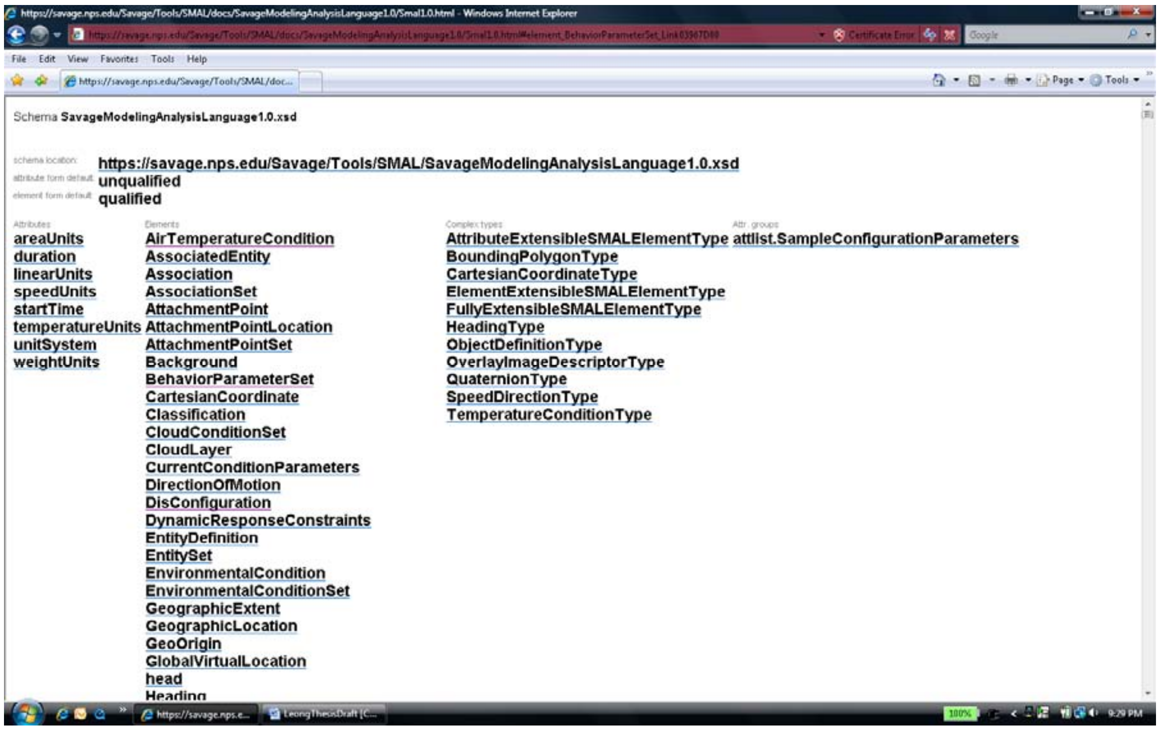
name	type	description
visualPerception	diskit.Sensor	Sensor object that processes all targets in the state space before ...
waypointCreator	diskit.WaypointCreator	The utility that creates waypoints based on the arguments passed
targetQueue	diskit.TargetQueue	The queue of active targets for this entity

Figure 10. VISKIT event graph editor is used to create event graphs. VISKIT autogenerates java source code and XML representation from the event graphs.

3. SAVAGE Modeling and Analysis Language (SMAL)

The SAVAGE Modeling Analysis Language (SMAL) is a XML based language providing comprehensive constructs for describing tactical, physical and simulation oriented metadata for vehicles, terrain and other entities in a virtual environment (Rauch, 2006). An excerpt from the SMAL online documentation is shown in Figure 11.

SMAL is used in Viskit, the SavageStudio scenario-authoring tool, the Scenario Authoring and Visualization for Advanced Graphical Environment (SAVAGE) model archives and the Savage Defense X3D model archives. The SMAL construct (specifically SimulationAgent element) in a Savage X3D model archive is the basis for matching DES behavior to X3D visualization.


https://savage.nps.edu/Savage/Tools/SMAL/docs/SavageModelingAnalysisLanguage1.0/Smal1.0.html - Windows Internet Explorer

Schema **SavageModelingAnalysisLanguage1.0.xsd**

schema location:	**https://savage.nps.edu/Savage/Tools/SMAL/SavageModelingAnalysisLanguage1.0.xsd**
attribute form default:	**unqualified**
element form default:	**qualified**

Attributes	Elements	Complex types	Attr. groups
areaUnits	AirTemperatureCondition	AttributeExtensibleSMALElementType	attlist.SampleConfigurationParameters
duration	AssociatedEntity	BoundingPolygonType	
linearUnits	Association	CartesianCoordinateType	
speedUnits	AssociationSet	ElementExtensibleSMALElementType	
startTime	AttachmentPoint	FullyExtensibleSMALElementType	
temperatureUnits	AttachmentPointLocation	HeadingType	
unitSystem	AttachmentPointSet	ObjectDefinitionType	
weightUnits	Background	OverlayImageDescriptorType	
	BehaviorParameterSet	QuaternionType	
	CartesianCoordinate	SpeedDirectionType	
	Classification	TemperatureConditionType	
	CloudConditionSet		
	CloudLayer		
	CurrentConditionParameters		
	DirectionOfMotion		
	DisConfiguration		
	DynamicResponseConstraints		
	EntityDefinition		
	EntitySet		
	EnvironmentalCondition		
	EnvironmentalConditionSet		
	GeographicExtent		
	GeographicLocation		
	GeoOrigin		
	GlobalVirtualLocation		
	head		
	Heading		

Figure 11. SMAL documentation on SAVAGE website (https://savage.nps.edu/Savage/Tools/SMAL/docs/SavageModelingAnalysisLanguage1.0/Smal1.0.html).

4. Defense Model Archives

The model library is an open-source set of 3D models used for defense simulation. Bugs are tracked online. The resources are available at https://savage.nps.edu/Savage. The Defense model archives is a similar set of models used defense simulation. However, the access is only limited to U.S. citizens and government contractors only. Catalog builder software reviews the two-tier SAVAGE model file directories and files to read in camel-case directory, file names, embedded document meta values and SMAL metadata nodes to create a content catalog in XML. The catalog builder is written in Java. An XSLT stylesheet then reads the SAVAGE content catalog and creates the various HTML pages associated with the models. A build.xml project file invokes XSLT stylesheets and facilitates in the creation of zip archives, uploading files and other deployment tasks. Subversion (SVN) is the tool used for the version control of source code and models. The archive infrastructure is extensible, with X3D-Edit and Netbeans both supporting model contribution by authors.

N. NETBEANS 6.1 AND VERSION CONTROL

1. NetBeans Integrated Development Environment (IDE)

With support for UML features, NetBeans is an integrated development environment for software application analysis, design and implementation. One can easily construct UML diagrams by selecting the design UML components required from the component palette and then dragging and dropping the components into the diagram editor. UML diagrams supported by NetBeans are use case, state, sequence, class, deployment, component, collaboration and activity diagrams, which provides good coverage for most software engineering needs.

The “apply design patterns” feature invokes a wizard to facilitate an intuitive step-by-step approach that incorporates UML design pattern templates into the software application design. Basic understanding of UML and design patterns is still necessary for effective use of the tools when building a quality software design. This is because it is the use case and its realization that determine the suitable design pattern to meet the original design requirement. “Over applying” a design pattern when there is no design requirement to support the design decision may result in problems such as reduced performance. The UML diagrams are useful for documentation purpose. Besides that, NetBeans UML diagrams are also the basis for forward engineering in NetBeans.

2. Forward and Reverse Engineering

NetBeans forward engineering refers to auto generation of source code from UML diagrams within NetBeans. The source code for packages and class diagrams (coupled with well-defined operations) can then be autogenerated. The source code for sequence of class method calls between class instances that are captured in UML sequence diagrams cannot be autogenerated. In other words, NetBeans only generates classes, their attributes and methods (without detailed implementation). Developers have to implement the details of each class method.

NetBeans reverse engineering refers to the autogeneration of design models by NetBeans from source code. This feature is especially useful for existing software applications that require design documentation updates. Source code can be imported into

NetBeans and then the basic design entities and models can be autogenerated from the source code. Each basic design entity represents packages or classes (with details on attributes and methods). These basic design models are the basic building blocks for developers to create various UML diagrams. NetBeans reverse engineering is also useful for easy synchronization of design models, given updated source code.

3. Implementation using NetBeans

Implementation is made easy in NetBeans with the use of the project creation wizard. The type of project supported by NetBeans includes java, web, enterprise, mobility, UML, SOA, ruby, C/C++, NetBeans modules and solution templates. The project creation wizard increases the productivity of the developer by providing basic software infrastructure for specific types of software applications. Hence, the developer only needs to focus on the higher level business logic implementation. NetBeans has intuitive icons to reflect the types of NetBeans components or entities created. The customizable multi-panels and dashboards also make development more efficient because they reduce memory load for developers to keep track of important information from different panels. The refactor feature in NetBeans is powerful and helps developers to make consistent changes across the application by consistently updating all dependent modules that require changes.

In application development, developers often work in a team. Although NetBeans supports version control tools such as CVS or SVN that helps to control source code changes contributed by different developers, collaboration usually includes more comprehensive requirements. Conventional collaboration tools such as groove or MSN typically include features such as file sharing, chatting, co-editing design documents, notification, forum discussion, etc. Such collaboration features can be incorporated into NetBeans. These emerging features will further enhance the capability of the application.

4. Subversion (SVN) for Version Control in NetBeans

SVN is a version control tool released by CollabNet Inc. in 2000. It is used for managing current and historical versions of source code and documents. It is released under the Apache License and is used by many open source projects. Integration of SVN as part of the NetBeans integrated development platform can eases the process of updating source code into common code repository. It also helps to synchronize and ensure that every developer in the development team adds or amends the right versions of source code. SVN features include commits (from client to code repository), branching (for parallel development), tagging (to synchronize release versions across software applications), check out (from code repository to client), diff (to check for differences in code between versions), etc. One of the reasons why SVN is preferred over CVS is because the SVN executes atomic commits of source code. Either the whole transaction is committed or none of it is committed. This mitigates the risk of partial commitment of source code when there is some system or infrastructure issue (for example, the network goes down during the source code commit process). Other advantages of SVN include versioning of directories, renaming, better performance, etc.

O. PROTÉGÉ AUTHORING TOOL FOR SEMANTIC WEB DOCUMENTS

Protégé is developed by Stanford University. It is a free, open source tool that provides end users with the means to create, visualize and update ontologies. The Protégé OWL editor enables users to build ontologies based on OWL. Protégé is extensible and has plug-ins architecture. Using Protégé Java APIs, developers can add and integrate application plug-ins into the Protégé platform to extend platform capabilities. One such plug-in is the OWL-S plug in developed by SRI International. SRI International is an independent, nonprofit research institute that conducts contract research for government agencies and businesses. The OWL-S plug-in provides a usable means for user to construct OWL-S for semantic web services. It has WSDL import feature that automatically establishes service profile, grounding and atomic processes. User can then add or edit OWL-S description. There are even graphical representation for OWL-S features such as services ontology establishment (profile, grounding, etc.) and composite processes. The XML for OWL-S is made transparent to Protégé users and is dynamically generated by Protégé as the user creates or updates using the OWL-S plug-in.

P. OTHER TOOLS AND SERVICES

GeoServer (Figure 12) is an open-source server that connects information to the geospatial web (http://geoserver.org/display/GEOS/Welcome). It is a type of Geographic Information System (GIS). Using GeoServer, one can publish, subscribe and edit information using open source standards. One has full control over the look of the map. Web Map Service (WMS) displays geographic data as raster images. Web Feature Service (WFS) communicates real geographic data to and from the user in the form of Geography Markup Language (GML). Web Map Service-Transactional (WFS-T) allows users to edit geographic data in transaction blocks. GeoServer supports WFS-T and WMS open protocols from OGC to produce JPEG, PNG, SVG, KML/KMZ, GML, PDF, Shapfiles, etc. GeoServer supports display of maps on web pages, UDig, GVSig, Google Earth and others. Each has the UI interactive capability such as zooming and panning.

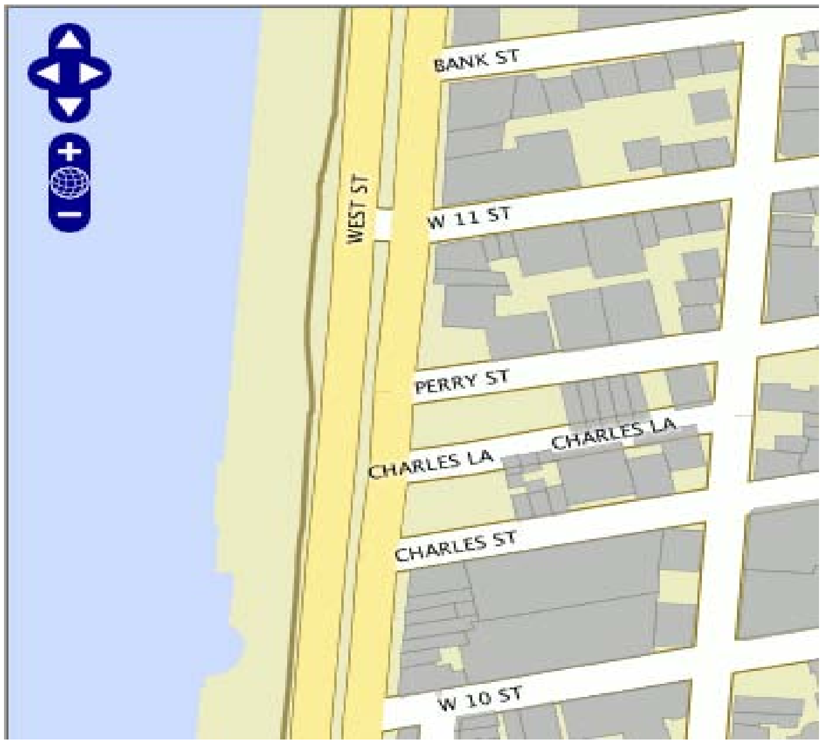


Figure 12. An example of GeoServer display.

Q. SUMMARY

The chapter has discussed the related background work. Software architecture considerations were discussed. WSA and web services are one realization of SOA.

Semantic web services leverage semantic web technologies and the technology further enhances the realization of SOA. Related technologies such as WSBPEL, WS-CDL and WS-Security are essential enablers for the WSAIF. SAVAGE-related tools and resources were discussed because they provide a domain for application of the WSAIF. The chapter concluded with descriptions of tools used in the implementation, testing and deployment of SAVAGE web services, WSBPEL and OWL-S models for purposes of this thesis.

III. ENVISIONED SOFTWARE ARCHITECTURE AND INTEGRATION

A. INTRODUCTION

This chapter discusses the three stakeholders of data, namely data owner, business process owner and software developer. The overview of software integration technology is elaborated. The speculated future software integration technology is mentioned. This is followed by the introduction to the concept of “smart” integration. The SOA design principles are discussed. The high level functional and architectural requirements for the envisioned WSAIF are elaborated. The WSAIF components realize the high level functional and architectural requirements.

B. STAKEHOLDERS

Data is the most important consideration when Information Technology (IT) initiatives such as hardware migration, software upgrade and revamp are applied to legacy software applications. These software applications contain critical data which are essential for core business functions and day-to-day operations. Hence, legacy software applications cannot be decommissioned or retired easily because critical data are tightly coupled with legacy software applications. Data formats can be designed specifically to support interoperability with legacy software applications. In order for legacy software applications to integrate with other software applications, it is necessary to migrate critical data to another format. Typically, an open format is adopted by all applications. However, the technical cost and operational risks for such an approach are high. Research performed by the Standish Group (2003) reports that 50-80% of a corporation’s IT budget is spent on maintenance. Hence, it is important to understand the evolution of software and the fundamentals of software integration (Hammer and Timmerman, 2008). In other words, a good understanding of the benefits, limitations, differences and related problems between different generations of software integration technologies can influence the design of a more efficient and maintainable future software integration framework.

Three stakeholders need to be considered (Figure 13): the data owner, business process owner and software developer (Hammer and Timmerman, 2008). The data owner is a person, commonly referred to as database administrator, who has full access to the data and knows what each data value refers to. The data owner maintains the data dictionary. The software developer is responsible for developing software applications and maintaining them in production. The tasks for a software developers can be outsourced to contractors. Thus, the software developer is often given limited access to the data. Business process owners are typically a manager or executives who understand the business value of the data, and knows how to interrelate the data to support business requirements.

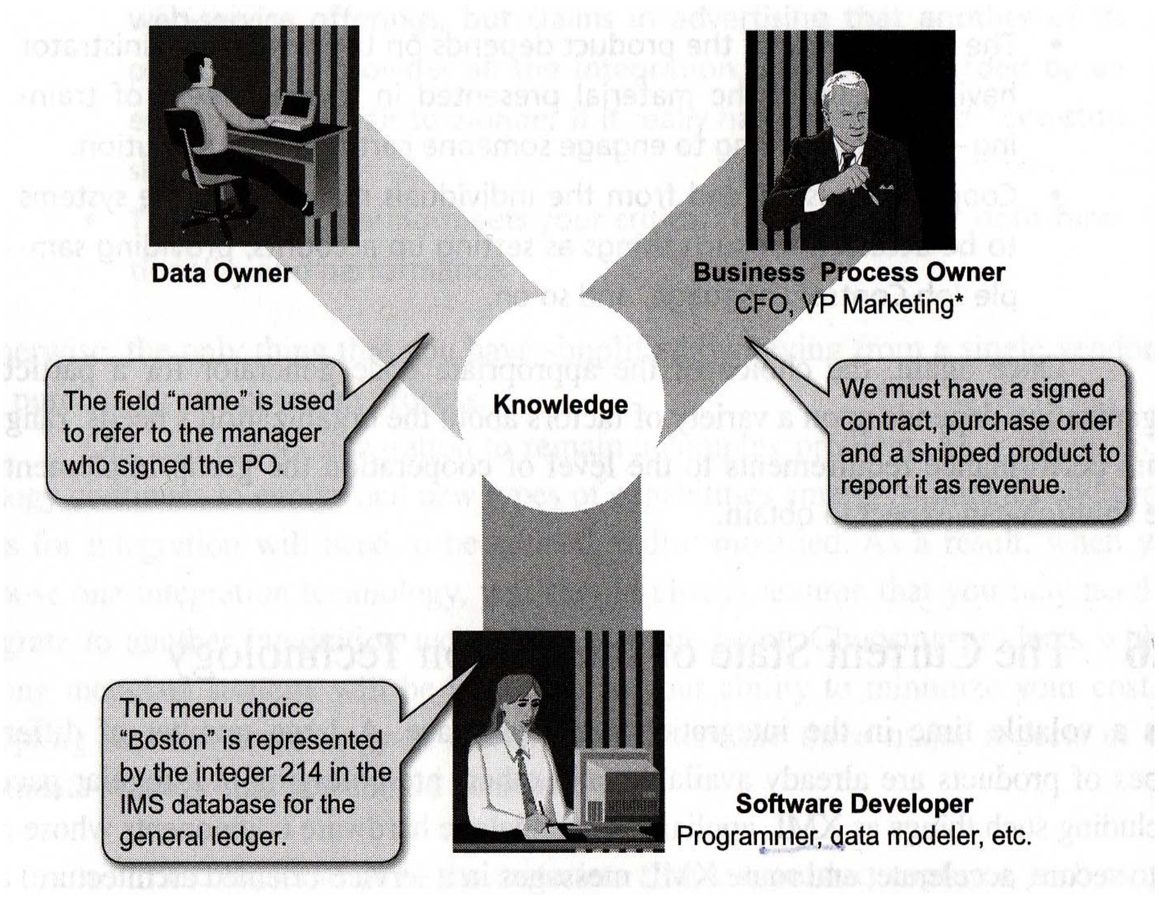

Data Owner
Business Process Owner
CFO, VP Marketing*
Knowledge
The field "name" is used to refer to the manager who signed the PO.
We must have a signed contract, purchase order and a shipped product to report it as revenue.
The menu choice "Boston" is represented by the integer 214 in the IMS database for the general ledger.
Software Developer
Programmer, data modeler, etc.

Figure 13. Three stakeholders for business knowledge and data (From Hammer and Timmerman, 2008).

C. SOFTWARE INTEGRATION TECHNOLOGY OVERVIEW

The following information on software-integration trends (database system, network system, desktop system, internet and code generators) is summarized and taken from Fundamentals of Software Integration (Hammer and Timmerman, 2008).

1. Age of Database Systems

Data integration is one of the key strategies for software application integration. The motivation is to ensure interoperability to a common data format so that different software applications can access the information. During the 1980s and 1990s, when relational database technology became commercially viable, research initiatives explored the possibility of integrating different types of database using relational database systems. There were two approaches to such a task. One approach was to design a hybrid or heterogeneous database management system (which is itself a relational database) that enabled hierarchical, network and relational databases to work together (Figure 14). The second approach was to develop productivity tools to automate migration of hierarchical and network-based databases into a single relational database. The latter approach was more successful. This was because the formal approach has issues such as complexity in configuration, compromised integrity of heterogeneous database transactions and high cost in maintenance.

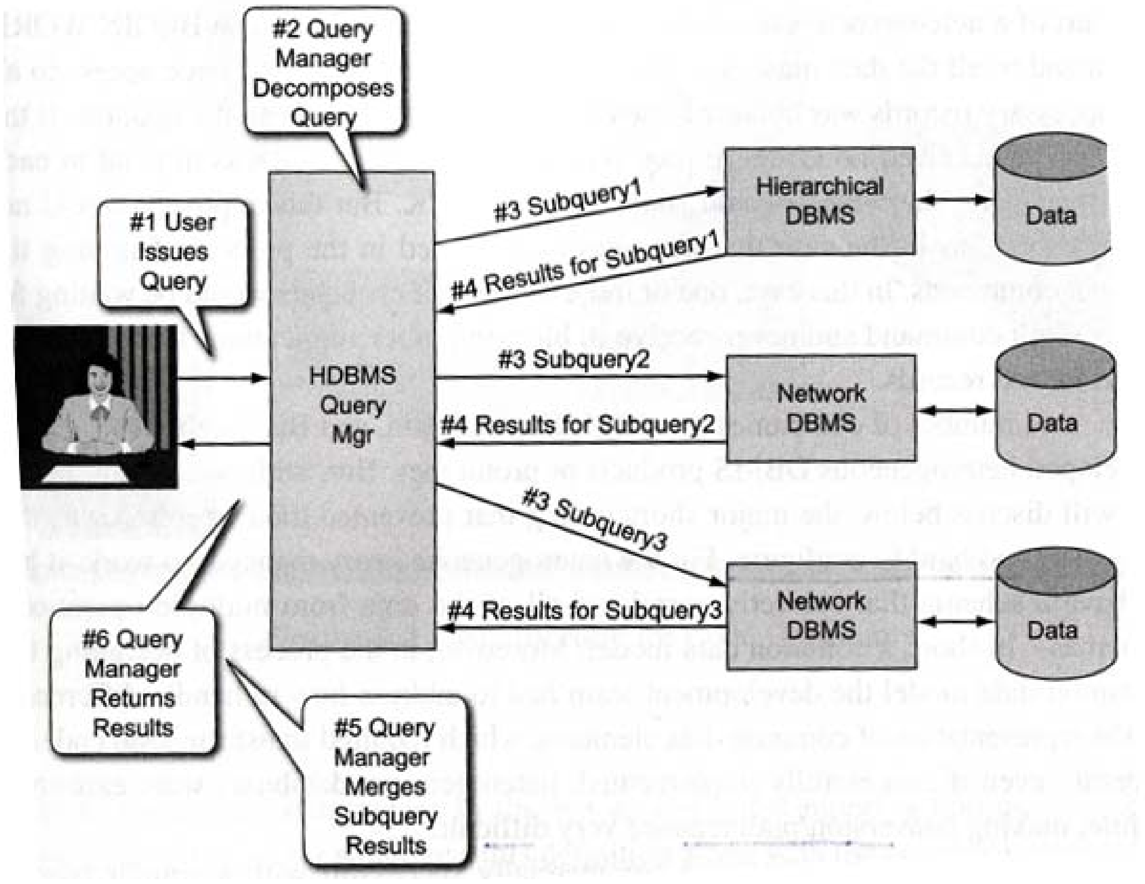


Figure 14. Heterogeneous DBMS enables hierarchical, network and relational databases to work together (From Hammer and Timmerman, 2008).

Code generators for databases were another software capability sometimes used for data integration. In this approach, hierarchical and network-based databases were migrated to a relational database using COBOL report writers. Code generators for databases take three types of input (Figure 15); schema for source database, schema for target relational database, and a control file which would indicate rules for correlating and transforming data. The output was source code for conversions. However, the generated source code was typically quite hard to read and thus difficult to troubleshoot and maintain. Metadata capabilities were used to describe relationships between source and target systems.

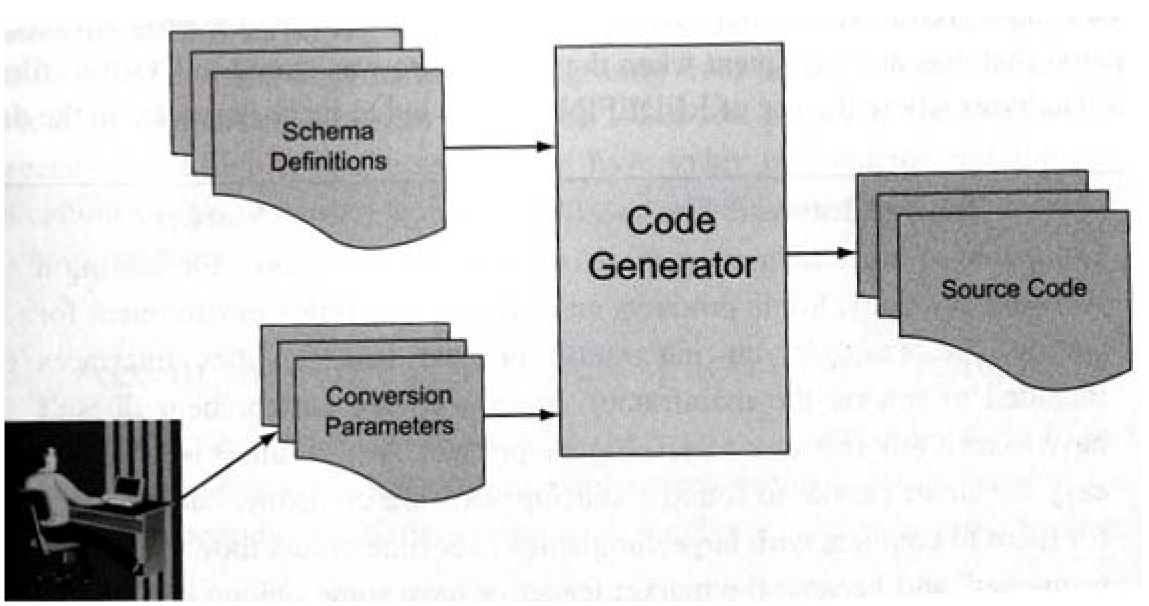


Figure 15. Code generators for database takes in schema for source database, schema for target relational database and control file. The output is source code (From Hammer and Timmerman, 2008).

2. Age of Network Systems

Creating a data warehousing was another solution for organizations to store and manage multiple databases. It was used effectively in a distributed environment. Its intended purpose was to facilitate analysis and reporting. As such, the technology strategy hopefully met management requirements and organizational needs. However, implementing and deploying warehouse solutions presented the follow challenges: 1) the need to understand the nature of data available in source system; 2) the need to specify the mapping logic between fields in data sources and attributes in warehouse schema; 3) the consideration for initial loading of data warehouse; and 4) refresh contents of data warehouse. Thus, various technological strategies were required to meet these multiple challenges.

Extract, transform and load (ETL) products provided database warehouse developers with a graphical user interfaces (GUI) to configure mappings between source and target databases and transformation logic to be performed on data values before updating the data warehouse (Figure 16). ETL products utilized an embedded RDBMS to process data, thus they had database engine-based architecture. Advantages would include loose coupling of ETL components to promote ease of installation. The GUIs enhanced usability of the system and embedded engines enabled capabilities such as audit trail of transactions that passed through the engine. Trade-offs would include the potential of the ETL engine becoming a bottleneck and performance overhead in view of managing transactions to staging tables.

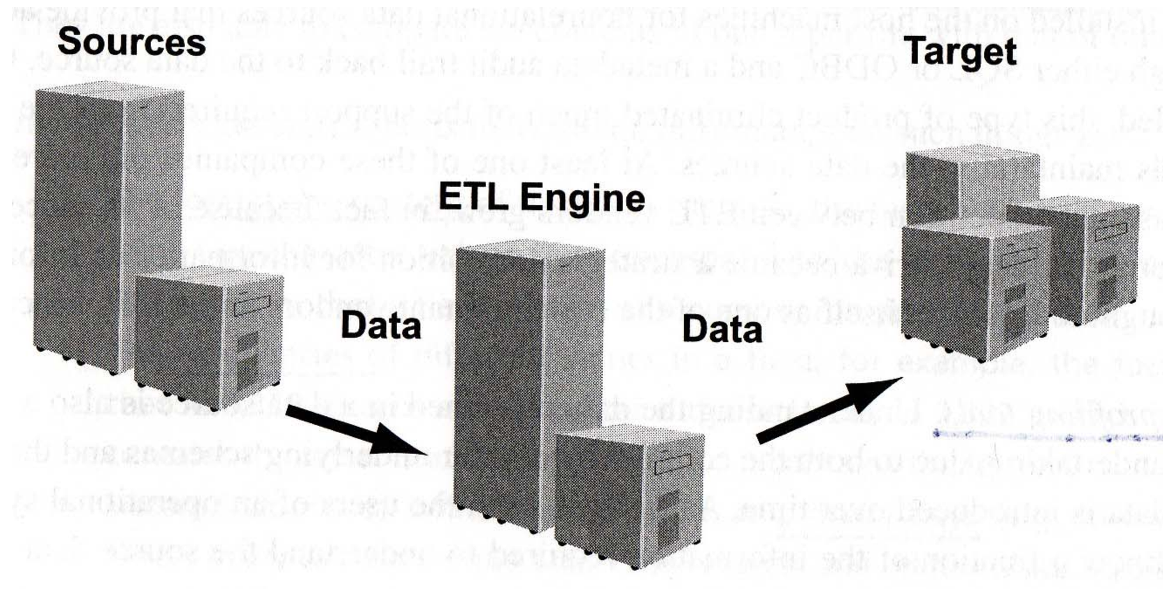


Figure 16. ETL architecture diagram. The deployment Engine could be a separate physical machine or running on the same machine (From Hammer and Timmerman, 2008).

Gateways provide relational interfaces to non-relational data sources. The technology was installed on servers with non-relational data sources deployed. It provided access to these data sources via Structured Query Language (SQL) or Open Database Connectivity (ODBC) and a metadata audit trail back to the data sources. Hence, such adaptation capability complemented the ETL engine.

Data profiling tools scan and analyze relational databases. They can operate with or without formal definition of input tables and were able to predict field boundaries by patterns in data. The tools generated reports that contain format and content of fields, frequency counts of values in data fields, and primary keys.

Data quality tools contained rich algorithms for “fuzzy matching” and entity resolution. For example, they were able to determine if two sets of data values refer to the same entity. Hence, they were good for tasks such as standardizing names and addresses, eliminating duplicate records, and determining household relationships.

In a data warehouse, metadata was used to represent its summary information. In other words, metadata provided the consolidated values about groups of transactions rather than the value of a single transaction. Challenges faced were to ensure metadata was updated or accurate, the limitations of not capturing the actual fields, and the time-consuming process to hand-code test and transformation logic.

Enterprise Application Integration (EAI) was defined as the use of software architecture principles to integrate a set of enterprise software applications. EAI leveraged methodologies and integration solutions such as object oriented programming (OOP). Middleware technologies such as Common Object Request Broker Architecture (CORBA), COM+, etc were possible enablers. Asynchronous message invocations were enabled by message queuing. Data standards were defined with XML. In order for EAI to be efficiently maintained, data interfaces that translated application-native API format data interfaces to a standard message format in XML (or vice versa) need to be defined.

Adaptor technology emerged as software vendors offer software adaptors to ease integration of Commercial Off-The-Shelf (COTS) applications. There were two types of adapter technology. Plug-and-play adapters automated transaction-level integration across various COTS applications. Each adapter translated between application-specific formats to a neutral format supported by middleware. As such, adaptation was point-to-point and application-specific. This results in maintainability and test complexity issues when the adaptation strategy was scaled to a large organization’s requirements. Another type of adapter was standards-based adapters (Figure 17). Basically, the adapters provided access to multiple sources by using a standards-based API (ANSI SQL-92). According to Hammer and Timmerman, the adapters were less automated but more successful when compared to plug and play adapters. The prerequisites for standards-based adapters are standardized formats for data exchange.

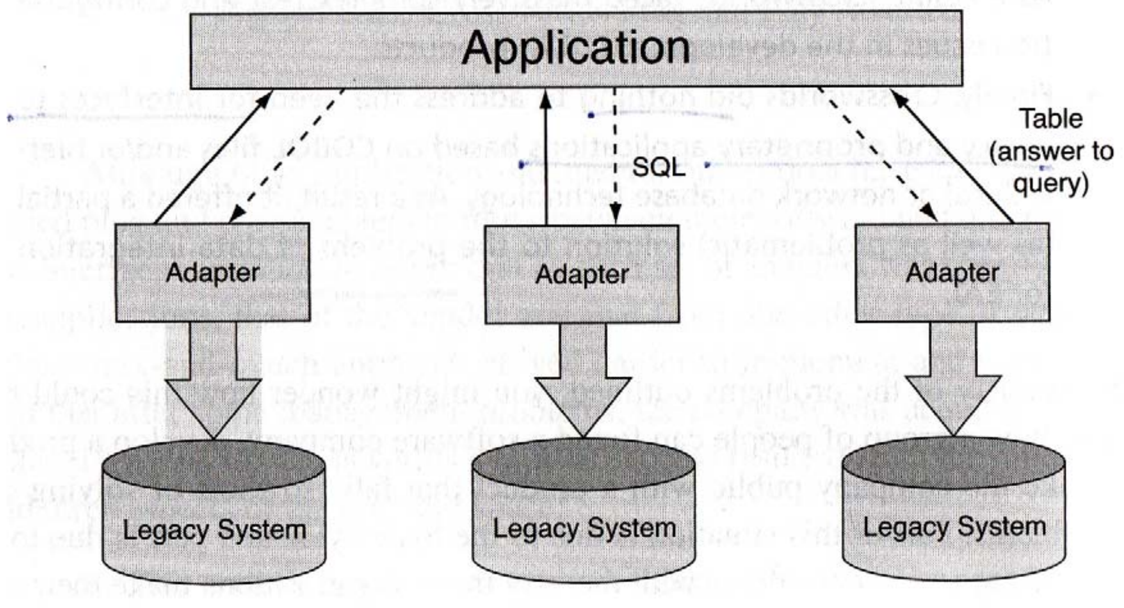


Figure 17. Standards-based adapters provide access to multiple sources by using standards-based API (From Hammer and Timmerman, 2008).

3. Age of Desktop Systems

Software applications such as word processers and spreadsheets run on desktop systems. These became popular and widely accepted because they met the business needs and requirements as a productivity tool for working professionals. Basic IT was a prerequisite skill for any office worker. Unlike servers which were housed in secured server rooms, desktop systems are more prevalent. They operated as client end terminals and are not as secure as servers. Desktop users stored information on the local machine. Hence, relaxed controls on desktop machines could introduce security risks such as compromising classified information transfer. Configuration management issues such as using out-of-date documents was also possible.

4. Age of Internet

Web-based applications are often more successful than custom-connected systems. This is because web-based applications enabled efficient search of large amounts of information for specific documents and information of interest, mechanisms for collaboration between individuals on different parts of a network, online shopping purchases and remote technical support. On the other hand, web-based applications also increased the complexity of integration. This is due to the increase in data types such as text, audio, graphics and video. The boundaries of working groups had thus widened to include customers, partners and even hackers. There was also increased security risk in view of the vulnerabilities of open data and protocol standards such as XML and web services invocation via HTTP bindings. Common security implications are important to all web-based applications that are open standards compliant.

Enterprise Content Management (ECM) helps organizations keep track of documents and provided an audit trail on the handling of each document. ECM had improved to incorporate different types of content such as web pages, graphics, videos, etc. Hence, ECM provides a means to manage and integrate different types of data, an important capability for web-based applications.

Enterprise Information Integration (EII) is the process of integrating information by providing data abstraction to a large set of heterogeneous databases. Hence, the user only saw a single interface. SOA established an elegant style and sound principles of designing, managing and implementing business modules as services in a distributed network. However, there was a need for middleware to realize the mediation and execution within a runtime environment that correlated a user's logical or business view of the problem space with backend data sources. EII may be one such enabler.

With metadata, users were empowered to make more informed decisions. Through research and development, software integration was becaming more efficient and secure. As software integration became a key competence within software industries, vendors were becoming more aware of the importance of metadata strategies for software integration. Thus the next challenge would be the need to assess the quality and completeness of metadata and the amount of sharable metadata on the internet.

5. Code Generators for Integration

SOA and the success of many strategic applications were dependent upon an enterprise approach to integration. There were considerations for code generators as a preferred approach to conventional enterprise integration. Basically, this was because of increased productivity which results in reduced cost. To elaborate: productivity gains could be realized for code generator solutions in maintaining multiple runtimes, but with performance degradation due to increased network traffic and heavy engine computation. Secondly, productivity gains were realized when code generators were compared to hand-coding in conventional enterprise integration. There were four criteria used for judging code generators: extensibility with respect to data sources and functionality, degree to which reuse and rapid change cycle were supported, ease of use, and acceptable performance.

6. Current State of Integration Technology

XML-based applications are emerging. This is facilitated by a SOA-based framework and infrastructure. WSA is one realization and web services are the enabler for vendors to position their products as application service providers (ASP).

As integration technology continues to evolve and new capabilities emerge, it is important to consider technology's flexibility to adapt to future changes when a certain integration technology solution is selected or developed. Choosing or developing products with a strong metadata strategy can help to minimize the cost of adopting a superior technology at a later time. There are three considerations in the selection of metadata strategy: completeness of the metadata, its flexibility for query, and the flexibility for the metadata to be read in another environment.

Industry analysts recommend organizations have an "integration competency center" since integration is central to IT. Thus, with more funds directed at software integration research and development, initiatives to accelerate the implementation of more productive, usable and intelligent software-integration strategies and solutions will be possible.

D. FUTURE INTEGRATION TECHNOLOGY

So what does the study of software integration trend leads us to? The importance of metadata has been mentioned repeatedly through the evolution of integration technology. The age of network systems has made deployment of software applications and components on distributed, decentralized platforms feasible. This is a preferred migration approach for legacy systems because they are typically constrained by the resources owned by different agencies residing at different nodes in a distributed network.

EAI also highlights the importance of data interfaces and integration infrastructure to perform impact analysis. The complexity and necessity of adapter technology cannot be neglected. Data-quality tools leverage heuristic algorithms and Artificial Intelligence (AI) techniques to enhance data quality. The age of desktop also highlights industrial awareness of the security risks from deploying rich software applications on desktops. During the age of internet, EII-based solutions are identified as an enabler for mediating and correlating users' logical view of the problem space with backend data sources within an SOA environment. Given the large amount of information flooding into the internet, ECM is necessary. There are also some considerations of technology that can potentially increase the productivity of software integration, one of which is code generators. In the current trend of integration technology, open standard-based protocols and standards such as XML and web services are gaining momentum as the preferred industrial implementation of SOA solutions.

Future software integration technology will be data-centric. Data refers to vendor-specific metadata, data sources, content and business views of end users. Metadata may not be sufficiently expressive. Thus, semantic web concepts are a superior approach to express and represent data, entities and relationships. Data needs to be made available and sharable in the network. How effectively this propagates depends on the policy maker. Software integration technology is only a means to establish the infrastructure which is necessary to facilitate sharing of information.

Data has to be portable and agreeable in a format recognizable by all collaborating nodes within the network. Open standards such as XML will be widely adopted by industries to ensure data interoperability. Data interoperability and scalability will be independent of proprietary runtime environments, platforms and programming languages.

AI planning and software agents are the key enablers for refining and enhancing quality of data processing for end users by facilitating automation in a run-time environment.

Security will continue to be the key focus to enable a heterogeneous and open software infrastructure. Software integration technology needs to ensure confidentiality, integrity and availability of data.

Reliable, scalable, flexible and adaptable middleware that can ensure interoperability between rich metadata and software agents is the basis to realize the future software integration technology. Software reuse will continue to be a major design consideration of the future integration technology.

The future integration technology should also transcend and not be bounded by any software paradigm. For example, the query and implementation of query will not be dependent solely on relational databases. Relational database and query will still exist in view of legacy systems. Both adaptation and standardization approaches have to work together. Adaptation will be interim approach until recommended/approved standardization sets in. A consistent and robust methodology and framework will need to be established to ensure such a process is repeatable.

The development and management of future integration technology will be driven by methodology. There will be continuous attempts to classify, generalize and structure integration processes. The purpose of an integration methodology is to simplify a complex issue so that it is repeatable and its behavior predictable.

The application of human factors and human computer interaction principles is necessary to improve the usability of integration technology. It should also be part of evaluation and test criteria to ensure software quality. Inspiration can be drawn from cognitive science to enhance the usablility and intelligence of software architecture.

E. "SMART" INTEGRATION

The continual emphasis on realizing a usable and intelligent architecture will eventually lead to the idea of smart integration. Smart integration is an emerging methodology for software integration. It is about the future design and implementation of software integration technologies. The architectural analysis will need to consider the high-level design requirements and strategies for architecture usability and intelligence. These will then be realized as use cases and functionalities. A consistent and repeatable process that integrates and supports the use cases has to be established. As such, smart integration is part of an integral software integration process that involves both humans and technology. Finally, performance criteria used to evaluate integration processes and technology will be clearly defined. The evaluation and feedbacks will be part of the smart integration process.

At framework implementation, software integration should be managed and executed autonomously by software agents. Adaptation code required to link different software components together will be auto-generated. The associated unit test classes can also be auto-generated. The end-user is required to manage and facilitate integration at the business level. The constraints, rules and logic required to match make and adapt various components are well defined and supported by state-of-the-art data modeling techniques such as the semantic web concepts. End users are equipped with well designed user interface and AI decision support tools. Hence, data can be efficiently and effectively managed by end users. Low level coding, implementation, testing and deployment can be undertaken by systems. As such, engineers may not be required to perform hard core coding. Instead, the focus can be on the configuration of software adaptation components.

F. THE IMPLICATION TO SOA SOLUTIONS

1. SOA Design Principles

SOA strategically aligned itself as the next trend in software integration technology. There are eight categories of SOA design principles (Erl, 2008), as shown in Figure 18. WSAIF will be usable and intelligent. It is a realization of SOA. WSAIF will be an enabler for smart integration in the SOA paradigm. Principles that result in the implementation of specific service design characteristics include standardized service contract, service reusability, service autonomy, service statelessness and service discovery. Principles that shape and regulate the implementation of design characteristics include service loose coupling, service abstraction and service composability.

The purpose of the service contract is to ensure a consistent way to describe service capabilities and overall purpose of the service. The key idea of service reusability is for services to contain agnostic logic such that they can be reusable enterprise resources. This will increase business agility, realize an agnostic service model and service inventories. There are three types of planned reusability. Tactical reusability requires immediate implementation of services that meet a critical requirement. Targeted reusability refers to service implementation meeting immediate and near-future requirements. Complete reusability refers to service implementation with a complementary range of functionality. Service autonomy refers to the run-time and design-time autonomy of a single service. Run-time autonomy refers to the control over processing logic when the service is invoked. Design-time autonomy refers to the control over making changes to the service over its lifetime. A service is designed to be stateless. Having the service stateless greatly reduces the computational complexity in maintaining state information. Hence, this maximizes service scalability and performance. Service discovery helps to determine if a required business function is made available within the service inventory. Service loose coupling refers to minimal dependency between a service contract and consumers and between a service contract and its underlying implementation. The motivation for service abstraction is to publish only necessary information and avoid dissemination of redundant service information. The idea of service composability is to match make or assemble different services. This establishes a process to solve a larger problem.

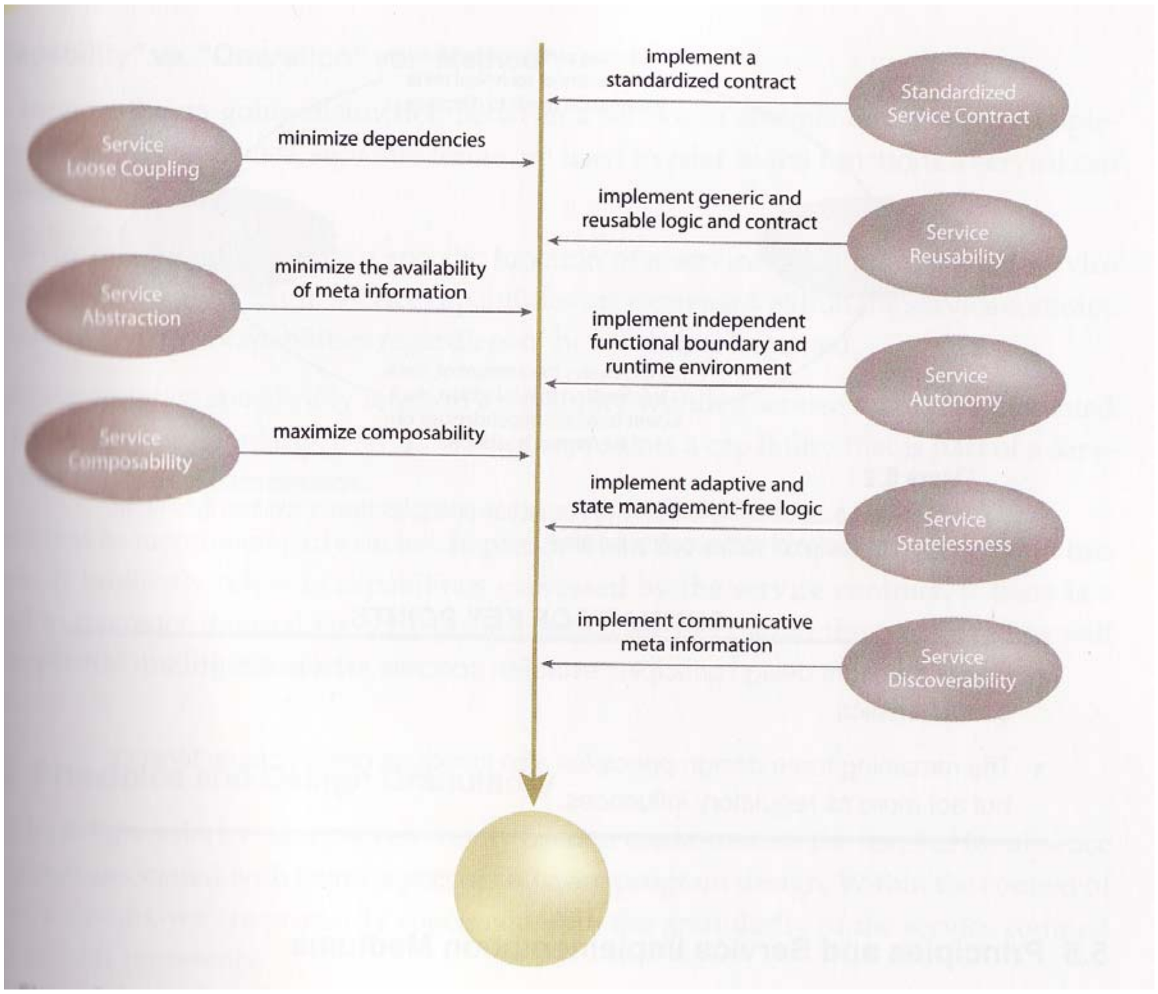

implement a standardized contract
Standardized Service Contract
Service Loose Coupling
minimize dependencies
implement generic and reusable logic and contract
Service Reusability
minimize the availability of meta information
Service Abstraction
implement independent functional boundary and runtime environment
Service Autonomy
Service Composability
maximize composability
implement adaptive and state management-free logic
Service Statelessness
implement communicative meta information
Service Discoverability

Figure 18. There are eight categories of SOA design principles (From Erl, 2008).

2. Envisioned WSAIF – A Realization of SOA

The envisioned Web Services Architecture Intelligent Framework (WSAIF) will be usable and intelligent. WSAIF is a realization of SOA. It handles the interoperability, facilitation, implementation and methodology for integrating the various SOA-based open standards. Interoperability refers to the well-defined component interfaces, model representation and protocol for cross-component interaction. Interoperability also takes into account a flexible approach to adapt SOA-related standards to WSAIF. Facilitation is the conceptual and logical sequence/order how the various WSAIF components work with each other. Implementation realizes and manages the run-time environment. It controls the deployment, invocation and recovery of the components within the run-time environment. Most importantly, the WSAIF needs a robust and consistent methodology to integrate and hold the components together. There are five major steps to the methodology: architectural analysis; identifying key areas of functionality; definition of strategy and process; planning for auditability and reuse; and criteria for evaluating integration technology (Hammer and Timmerman, 2008).

WSAIF will provide an Object Oriented abstraction that sits between the component interfaces and SOA-based open standards. WSAIF will be able to interoperate and perform tasks implemented by different open standards. Hence, WSAIF will be adaptable to SOA variations. For example, WSAIF can augment the process ontology defined in OWL-S with the adaptation rules defined in a WSBPEL *assign* element. The abstraction will also make the SOA variations transparent to the user of WSAIF. Hence, to the user, WSAIF is generic, flexible and usable.

WSAIF will be intelligent. It will exhibit autonomous behavior such as self-integration of web services, self-orchestration of business workflows and self-healing when web services deployment status changes. In other words, it will dynamically put together a comprehensive set of tools and data for a specific purpose and domain.

G. ENVISIONED WEB SERVICES ARCHITECTURE INTELLIGENT FRAMEWORK (WSAIF)

WSAIF needs to consider semantic web services architecture as a high level functional and architecture requirement (see Figure 19). This includes service discovery, service engagement (service contracting and negotiation), enactment and engagement (process monitoring, failure handling, dynamic composition, etc), community support services (common and reusable services) and quality of service.

1. WSAIF High Level Functional and Architecture Requirements

Service discovery – This high level functional requirement includes providers describing the identifiers, capabilities, queries, constraints, behavior, supported functions and abstract characterizations of offered services. Abstract characterizations of services are required for matchmaking purposes. Matchmakers compare the description of queries, capabilities, constraints and supported functions. Requesters verify that the discovered services meet the precondition requirement before using them. Architecture requirements include protocols for advertising and service discovery purposes.

Service engagement – This high level functional requirement includes the formulation of service requests, the basis for agreement, contracting preliminaries and contracting negotiation. Architecture requirements include protocols for negotiation and services to manage negotiation and auditing.

Service enactment and engagement – This high level functional requirement includes interpretation and translation of responses when the requester and provider use different ontologies for communication. With a good basis for choreography, interpretation and execution, the capability will result in higher quality dynamic service composition. The requirement for mediating and delegating processes that are composed is also important. Coupled with the mediating process is the process for status monitoring. In the event that an abnormality occurs, a notification is triggered. Service failure handling and compensation are required if the processes run into exceptions. There is also the need to resolve disputes and ensure compliance for services involving third-party tools. Requirements to ensure audit tracking, explanation, security and concurrency controls are important for SOA deployed in a multi-agency environment where providers and requesters for services can assume different roles and access. Architecture requirements include services for process mediation, scheduling, execution and composition. Status-logging and policy monitoring are also part of the architecture requirements.

Community Support Services – This high level functional requirement includes services for ontology lookup, mapping, version control, security, group membership, trust reasoning, community based preference and reliability reporting. It is also important to consider policy, protocol and lifecycle management services.

The quality of service (QoS) level agreement has to be defined. Considerations for QoS include deadlines, accuracy and cost. QoS has implications regarding how services are advertised, topics for negotiation processes, etc. There should be a means to monitor QoS and control the services accordingly.

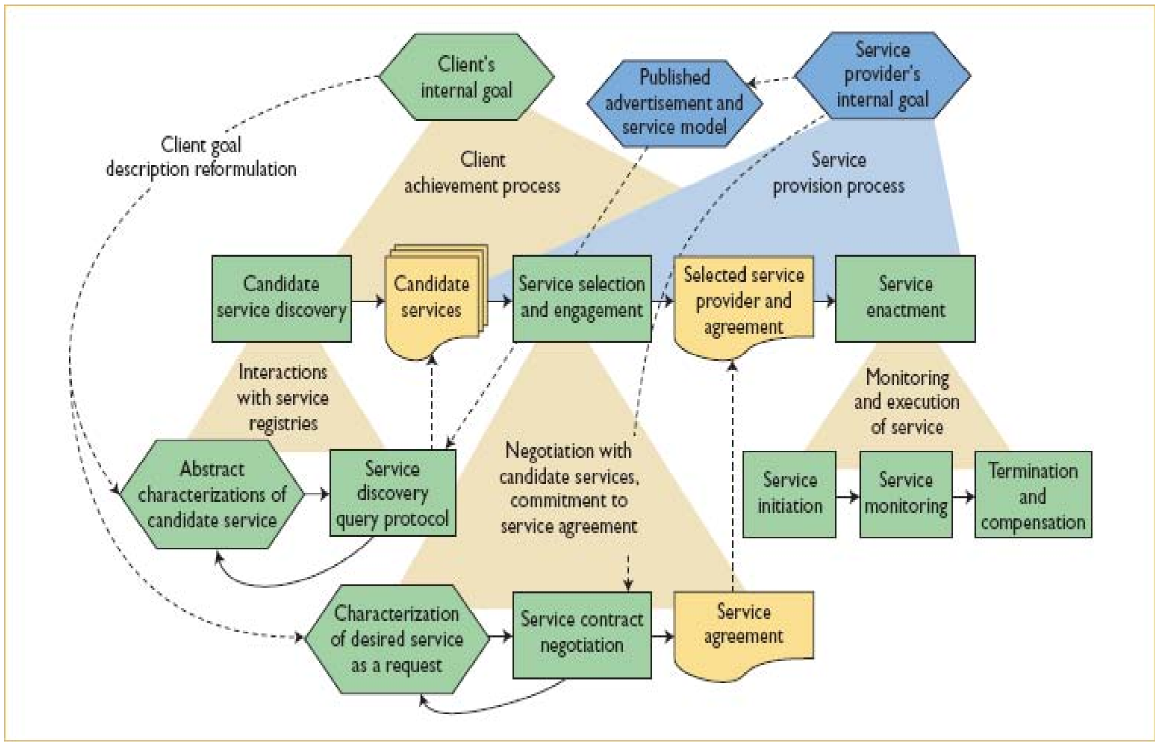

Client's internal goal
Published advertisement and service model
Service provider's internal goal
Client goal description reformulation
Client achievement process
Service provision process
Candidate service discovery
Candidate services
Service selection and engagement
Selected service provider and agreement
Service enactment
Interactions with service registries
Monitoring and execution of service
Abstract characterizations of candidate service
Service discovery query protocol
Negotiation with candidate services, commitment to service agreement
Service initiation
Service monitoring
Termination and compensation
Characterization of desired service as a request
Service contract negotiation
Service agreement

Figure 19. Semantic web service architecture high level functional and architecture requirements (From Burstein *et al.*, 2005).

Client (green) and service provider (blue) goal descriptions (hexagons) drive the three main phases of interaction (discovery, engagement, and enactment). At the lower level, these goals are communicated during message exchanges utilizing the protocols (green boxes) that follow general, phase-specific patterns (Bursteinet *al.*, 2005).

2. WSAIF Components

With the design considerations and capabilities mentioned above, WSAIF will be sufficiently intelligent to filter, identify and notify relevant, reliable and useful information to the user. Thus, WSAIF is usable from knowledge management perspective.

To realize the functional and architecture requirements, components of WSAIF include (Figure 20): orchestrator (elaborated in this paper in Section 6); matchmaker (to facilitate autonomous match-making of web services); agents (applied heuristic search for the right web services); adaptor (to address the heterogeneity problem between two web methods); communication (supporting asynchronous and synchronous processes);

security; choreography; and user interface. The WSAIF manager component is the main controller for the architecture framework. Each of WSAIF components will be elaborated below.

WSAIF Orchestrator – The purpose of the Orchestrator is to establish a comprehensive and consistent representation to model composite processes or workflows. The software framework establishes an abstraction between a common set of APIs that ensure interoperability between WSAIF components and the various open standards for modeling business processes such as OWL-S and WSBPEL.

WSAIF Matchmaker – The purpose of WSAIF Matchmaker is to facilitate the execution and search for suitable/matching services.

WSAIF Agents – The Agents component implements the various AI search algorithms. WSAIF agents assess the process scenario and execute the most suitable AI search algorithm. It recommends suitable services.

WSAIF Choreography – The Choreography component specifies the pattern or sequence of assets for a particular service. This information is an input to the WSAIF Matchmaker. The framework abstraction incorporates open standards such as WS-CDL.

WSAIF Adaptor – The Adaptor component handles the syntactic and semantic mismatches between parameters of atomic processes which form part of a composite process. The translation logic that “glues” two atomic processes is expressed in XSLT.

WSAIF Communication – The Communication component implements and supports synchronous and asynchronous communication between atomic processes. Synchronous communication simply means all parties involved in a communication have to be present at the same time. It is direct. Thus, a request invoked by a party would expect and wait for a response from the receiving party. An example of synchronous communication is chat. Asynchronous communication does not require all parties to be present at the same time. Thus, a request invoked by a party need not wait for a reply. An example of asynchronous communication is email. WSAIF Communication should support both wired and tactical wireless environment. Design strategies for overcoming limited bandwidth include incorporating Efficient XML Interchange (EXI) and lazy loading (i.e,. load in batch and only by need basis). Design strategies for overcoming limited connectivity include using robust asynchronous messaging framework.

WSAIF Security – This is an important component because services are deployed in a multiple-agencies environment. Providers and consumers of services can belong to different agencies. Each of them will have a specific role and authenticated resources. Hence, access control, authorization and authentication of web services are important. It is also important that the messages delivered from one agency to another need to preserve confidentiality and integrity. Thus, SSL and open standards such as XML-Security which includes XML encryption and XML signature are important realizations of such security design requirements.

WSAIF User Interface – The User Interface component works with other components and provides meaningful display of useful and important information. It also provides an intuitive means for users to employ the various functionalities in WSAIF. The WSAIF User Interface is flexible. An end user is able to configure and customize the specific UI components to be used for his dashboard. A UI mediator will synchronize and facilitate updates on dependent UI components when the information on one UI component changes. The WSAIF User interface will be built on a rich client platform. This is necessary to support more complex and dynamic UI interactions and controls. Furthermore, rich clients are able to cache and manage larger batches of information. Thus, it has the benefit to retain functionality on the client even when the network connection drops.

WSAIF Manager – The WSAIF Manager is the main/key controller for the architecture framework. WSAIF Manager understands the current process status and the abstract requests from WSAIF Orchestrator. WSAIF Manager then triggers WSAIF Matchmaker which takes in the input of WSAIF Choreography before formulating a detailed matchmaking request. WSAIF Matchmaker then sends the request to WSAIF Agent for execution. WSAIF Agent assesses the matchmaking request and search scenario. It selects the most suitable search algorithm, generates a search agent and performs search. WSAIF Agent then returns recommended services to WSAIF Manager. WSAIF Manager then sends the recommendation to WSAIF UI which then displays the result intuitively to the end users. End users interact with WSAIF UI and any updated information on business processes is sent to WSAIF Orchestrator.

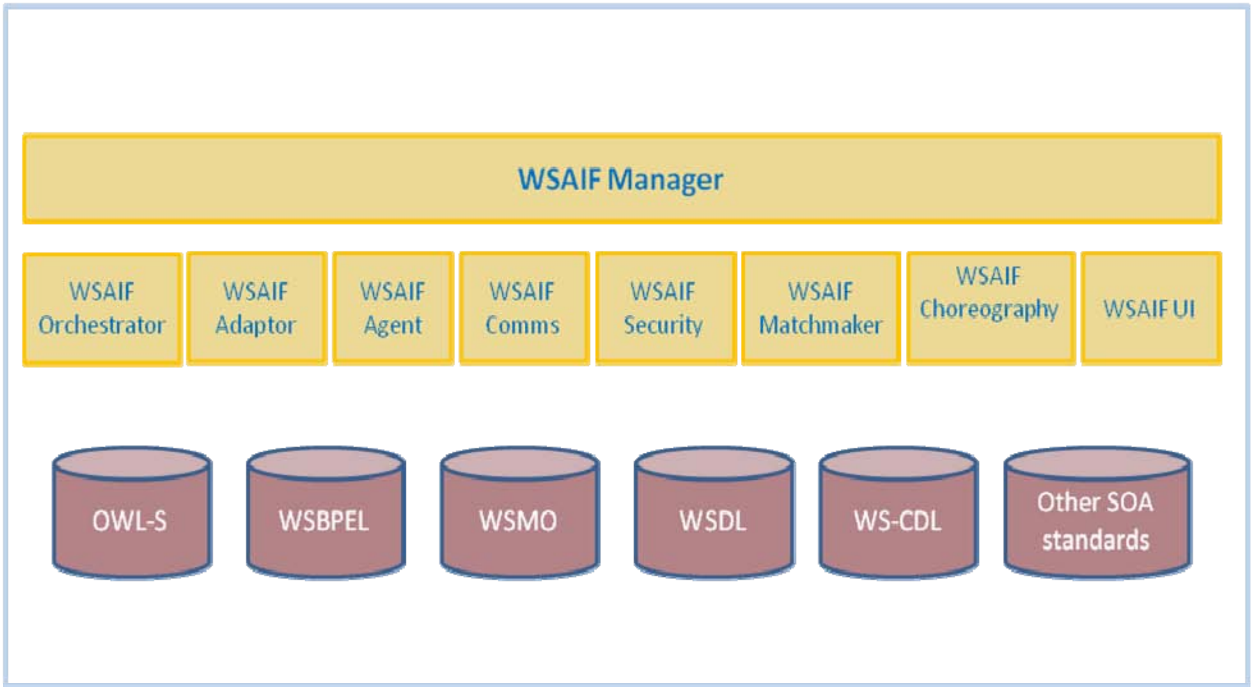


Figure 20. The envisioned WSAIF and its architecture components.

H. SUMMARY

The chapter has discussed the evolution of software integration technologies. Understanding the trends and associated failures or successes helps in the speculation of a future integration technology which is usable and intelligent. The emphasis on such attributes in future integration technology will eventually lead to smart integration approaches. The smart integration approach will be supported by a well-defined and robust methodology. To put it simply, future software integration approaches will no longer require software engineers to manually produce adapter code to ensure two software components interoperate. Rather, the approach is to design and implement user-friendly executive dashboard, configuration tools and intelligence into the software architecture so that software components can integrate without the need for hand-coding. In this thesis, SOA is used as the architecture paradigm to realize the concept. SOA design principles were reviewed in this chapter. The design principles will be the key design considerations for WSAIF, resulting in the high level functional and architecture requirements based on its association and relevance to semantic web service architecture. The chapter concluded with a detailed description of WSAIF software components required to address the functional and architecture requirements.

IV. SAVAGE WEB SERVICES

A. INTRODUCTION

This chapter first addresses the various implementation approaches for SAVAGE web services. The use cases are then discussed. The UML component diagram which realizes the use cases is elaborated. This is followed up by the details of class hierarchy for each component. The details of class object sequence interactions for each use case is also elaborated. The chapter concludes with description of the design approach to extend SAVAGE web services to incorporate new web methods.

B. USE CASES

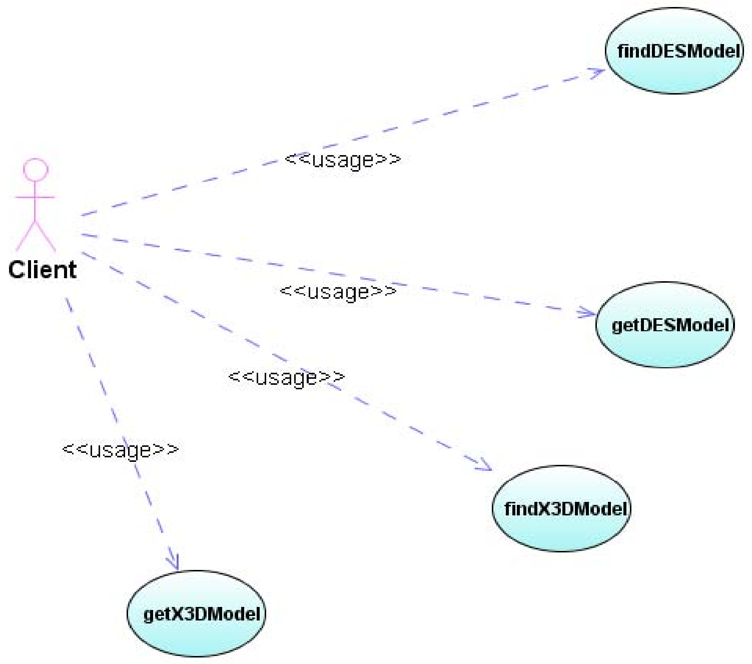


Figure 21. SAVAGE web services use case diagram in UML shows four use cases. The user uses the client to invoke *findX3DModel*, *getX3DModel*, *findDESModel* and *getDESModel* web methods.

There are two web services implemented for SAVAGE, namely *X3DWebService* and *DESWebService*. The web services are implemented in Java. “Client” refers to an external Java program which can run on a different machine. “Server” refers to the machine on which SAVAGE Web Services are deployed. For *X3DWebService*, the implemented web methods are *findX3DModel* and *getX3DModel*. For *DESWebService*, the implemented web methods are *findDESModel* and *getDESModel* web methods. The technical use case specification for each of the web methods are as follows:

1. ***findX3DModel*** **Web Method Use Case Specification**

The purpose of the *findX3DModel* web method is to find a list of matching X3D models given search term(s). The client invokes web method *findX3DModel* with parameter *searchTerm*. Parameter *searchTerm* is the keyword(s) of interest given by users. the client invokes the *findX3DModel* web method. The server receives the web service request from the client. The server iterates through page elements in the SAVAGE catalog and performs a string match between *searchTerm* and the content of name, title and description attributes. For the content of the description attribute, words are first tokenized before each tokenized word is compared with *searchTerm*. The server returns a list of matching X3D names and their associated URLs in XML format back to the client.

2. ***getX3DModel*** **Web Method Use Case Specification**

The purpose of the *getX3DModel* web method is to retrieve an X3D model file given its URL. The client invokes web method *getX3DModel* with parameter *URL*. *URL* refers to the unique resource locator for the X3D model file on the SAVAGE server. The server receives the web service request from the client. The server goes to a specific sub-directory as indicated by the URL. The server reads the X3D model file into a string and returns the content to the client.

3. ***findDESModel*** **Web Method Use Case Specification**

The purpose of the *findDESModel* web method is to find the matching DES behavior model given an X3D visualization model. The client invokes web method *findDESModel* with parameter *x3dURL*. The server receives the web service request from the client. The server iterates through page elements in the SAVAGE catalog and if there is a SMAL element defined (specifically *BehaviorParameterSet* child element), the server will retrieve information from the agent and URL attributes. This information is then returned to the client in XML format.

4. *getDESModel* Web Method Use Case Specification

The purpose of the *getDESModel* web method is to retrieve the DES behavior model in XML format from the SAVAGE repository. The client invokes web method *getDESModel* with parameter *desURL*. *desURL* will indicate the specific subdirectory where the DES model resides in the SAVAGE repository. The server receives the client's web service request, retrieves the DES model from the SAVAGE repository and returns the DES model to the client.

C. DESIGN CONSIDERATIONS

As discussed in Section 2, there are five key architecture considerations; namely, reliability, performance, scalability, security and maintainability. The scope and defined quality of SAVAGE web services design will be based on the five architecture considerations.

SAVAGE web services have to be maintainable. There will be distinct architecture layers (implemented via Java packages) to contain business logic and model related classes.

There is a need to consider possible future enhancements to include new web methods and associated business logic implementation. Hence, SAVAGE web services need to be extensible. Web service classes should be distinct in their roles and the name of web method should well describe its functions. The Strategy pattern is selected to create the layer of abstraction between the web service classes and the implemented strategy for the associated web method. This will decouple and enhance extensibility for the implementation of web methods and their associated business strategies.

Consumers of web services can be external parties within the collaboration network. Hence, as part of the maintainability consideration, it is important to ensure an intuitive way to assess the SAVAGE web services. No special XML format is required from consumers. Web services are invoked with simple input types. For example, the *findX3DModel* web method should take in the search term as the input string. It is not necessary for the input string to conform to any specific XML format in this case.

This is an experimentation or demonstration setup, hence SAVAGE web services are considered reliable if the invocations do not crash the system given the following conditions:

1. Small number of concurrent users (i.e., less than 5 users)
2. Support short demonstration period (i.e., less than 2 hours)

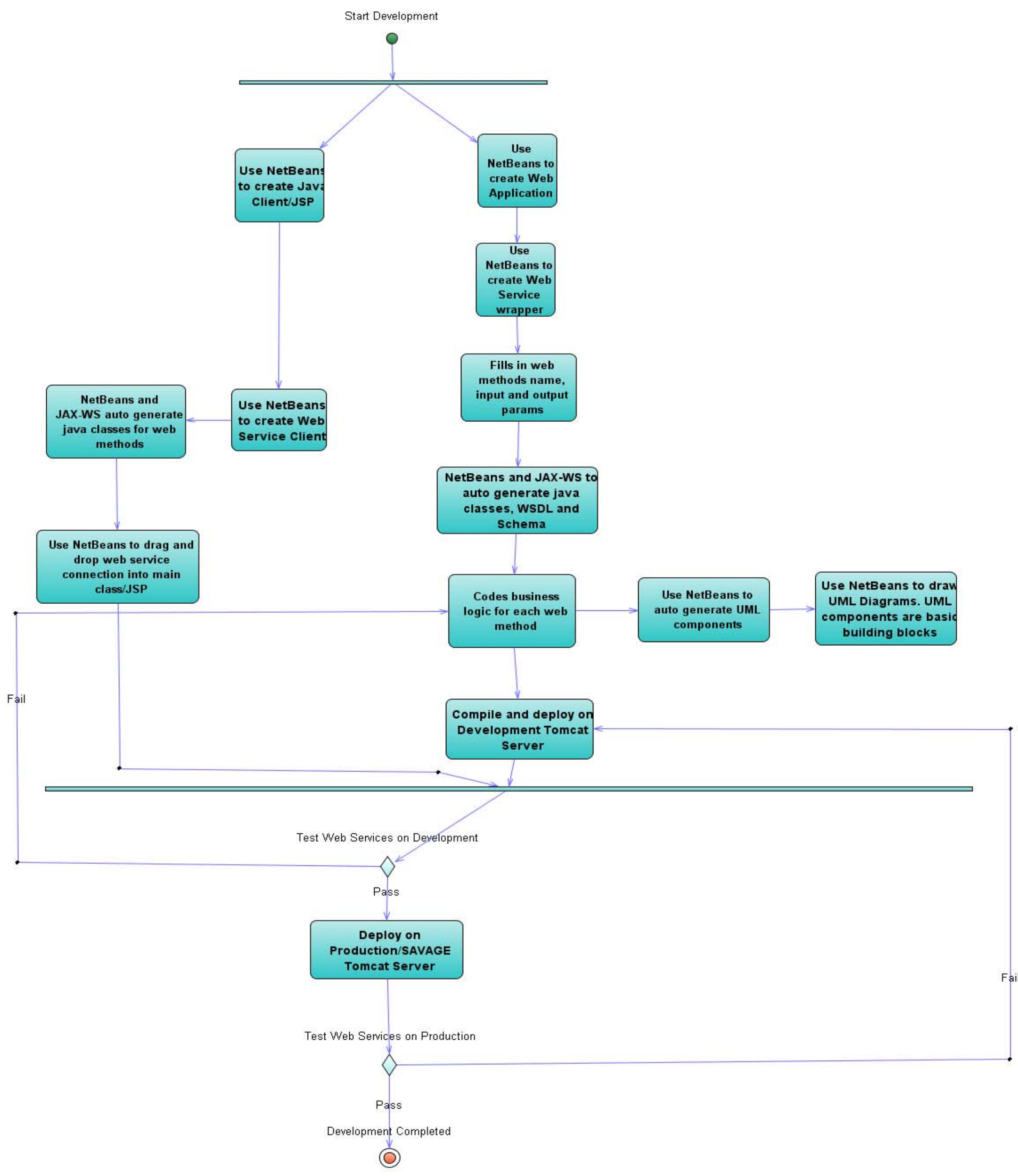
Thus, given the above design considerations, an enterprise solution which requires clustering of web services on multiple machines is not necessary.

The performance of the SAVAGE web services should be acceptable. Synchronous invocation of web services should be less than 5 seconds. If the synchronous web services call requires longer processing time, visual feedback should be implemented at the application user interface.

The scope of the SAVAGE web service implementation should include security consideration for communication confidentiality. Thus, transport level encryption such as SSL is necessary.

D. IMPLEMENTATION PROCESS

This section describes the development process for SAVAGE web services. Forward engineering using NetBeans provides the capability for NetBeans to auto-generate classes, attributes and methods (without business logic implementation) from UML diagrams. Reverse engineering (Figure 21) leverages available code to generate UML components, which are the basic building blocks for UML diagrams. NetBeans and JAX-WS auto-generate WSDL, schema and Java classes for the web methods. To establish web services operation hosted on a server, use the “new→web service” option in NetBeans. For client connectivity to web services, use the “new→web service client”. As such, establishing a web services connection can be done easily in NetBeans. Developer simply performs “drag and drop” operation on the web services client connection instances into Java main class or JSP code. Alternatively, the developer first develops WSDL using an XML editor such as XML-SPY. Then, WSDL can be imported into NetBeans using the “new→Web Services from WSDL” option. NetBeans and JAX-WS then read in information such as operations, endpoints and bindings from WSDL and auto-generate schema and Java classes for the web methods. The developer can then use NetBeans to perform forward or reverse engineering to create web services after importing WSDL.



Start Development
Use NetBeans to create Java Client/JSP
Use NetBeans to create Web Application
Use NetBeans to create Web Service wrapper
Fills in web methods name, input and output params
NetBeans and JAX-WS auto generate java classes for web methods
Use NetBeans to create Web Service Client
NetBeans and JAX-WS to auto generate java classes, WSDL and Schema
Use NetBeans to drag and drop web service connection into main class/JSP
Codes business logic for each web method
Use NetBeans to auto generate UML components
Use NetBeans to draw UML Diagrams. UML components are basic building blocks
Fail
Compile and deploy on Development Tomcat Server
Test Web Services on Development
Pass
Deploy on Production/SAVAGE Tomcat Server
Fail
Test Web Services on Production
Pass
Development Completed

Figure 22. Reverse engineering leverages available code and generates UML components. NetBeans and JAX-WS auto-generate WSDL, schema and Java classes for the web methods.

In the reverse engineering process, WSDL, Schema and Java classes for web service connectivity are auto-generated by NetBeans and JAX-WS.

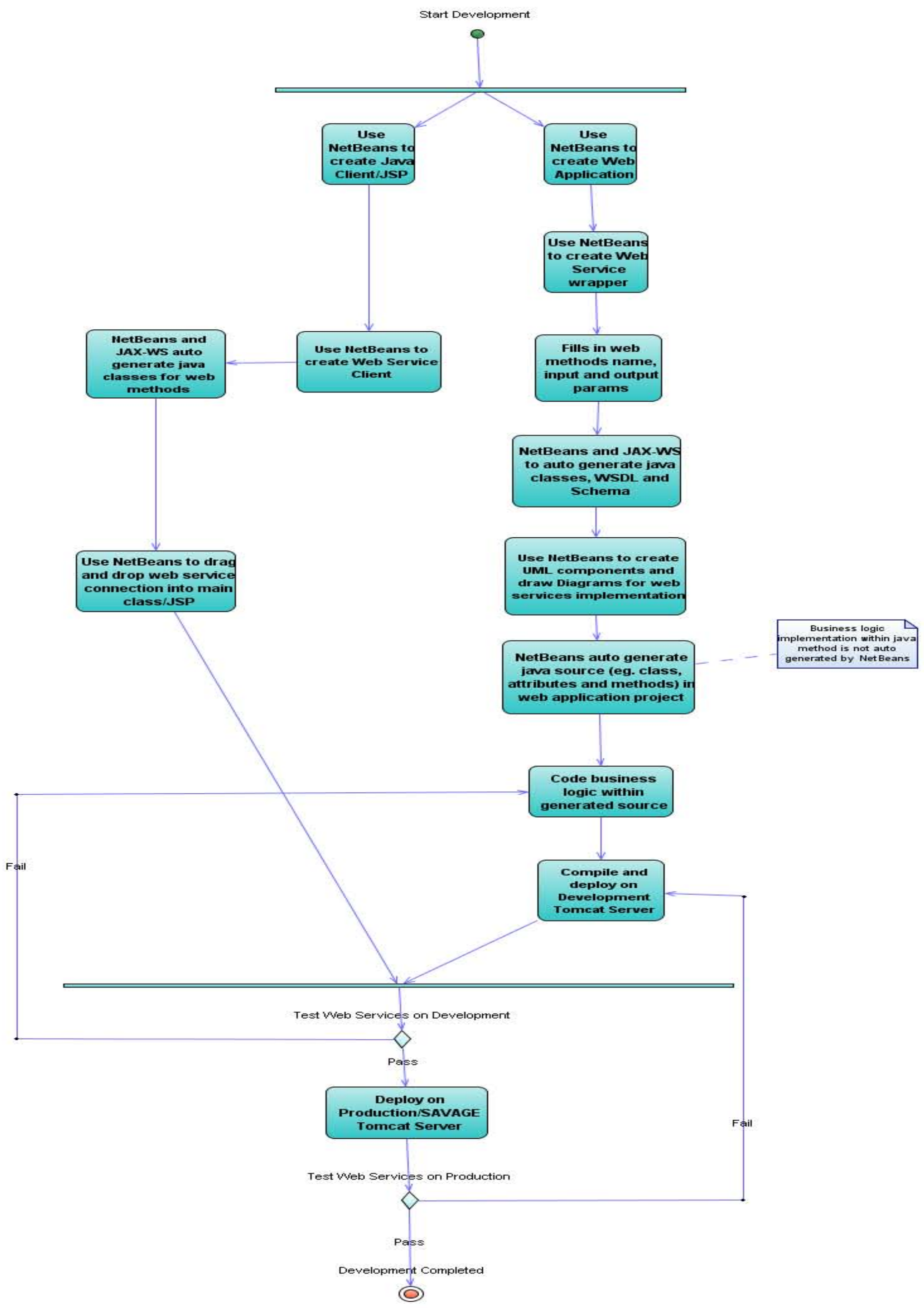

Start Development
Use NetBeans to create Java Client/JSP
Use NetBeans to create Web Application
Use NetBeans to create Web Service wrapper
NetBeans and JAX-WS auto generate java classes for web methods
Use NetBeans to create Web Service Client
Fills in web methods name, input and output params
NetBeans and JAX-WS to auto generate java classes, WSDL and Schema
Use NetBeans to drag and drop web service connection into main class/JSP
Use NetBeans to create UML components and draw Diagrams for web services implementation
Business logic implementation within java method is not auto generated by NetBeans
NetBeans auto generate java source (eg. class, attributes and methods) in web application project
Code business logic within generated source
Fail
Compile and deploy on Development Tomcat Server
Test Web Services on Development
Pass
Deploy on Production/SAVAGE Tomcat Server
Fail
Test Web Services on Production
Pass
Development Completed

Figure 23. Forward engineering using NetBeans provides the capability for NetBeans to auto-generate classes, attributes and methods (without business logic implementation) from UML diagrams.

For the forward engineering process, WSDL, schema, and Java classes for web service connectivity are auto-generated by NetBeans and JAX-WS. The web services implementation, consisting of Java classes with their class hierarchy, class attributes and methods (without business logic implementation), can be auto-generated from the UML diagrams. Details are described in the UML activity diagram shown in Figure 22.

WSDL can be configured separately by using an XML editor. The pre-configured WSDL can be imported into NetBeans by selecting option “new→web service from WSDL”. The developer can then perform the remaining steps in the forward or reverse engineering processes using NetBeans. Details are illustrated in the UML activity diagrams in Figures 22 and 23.

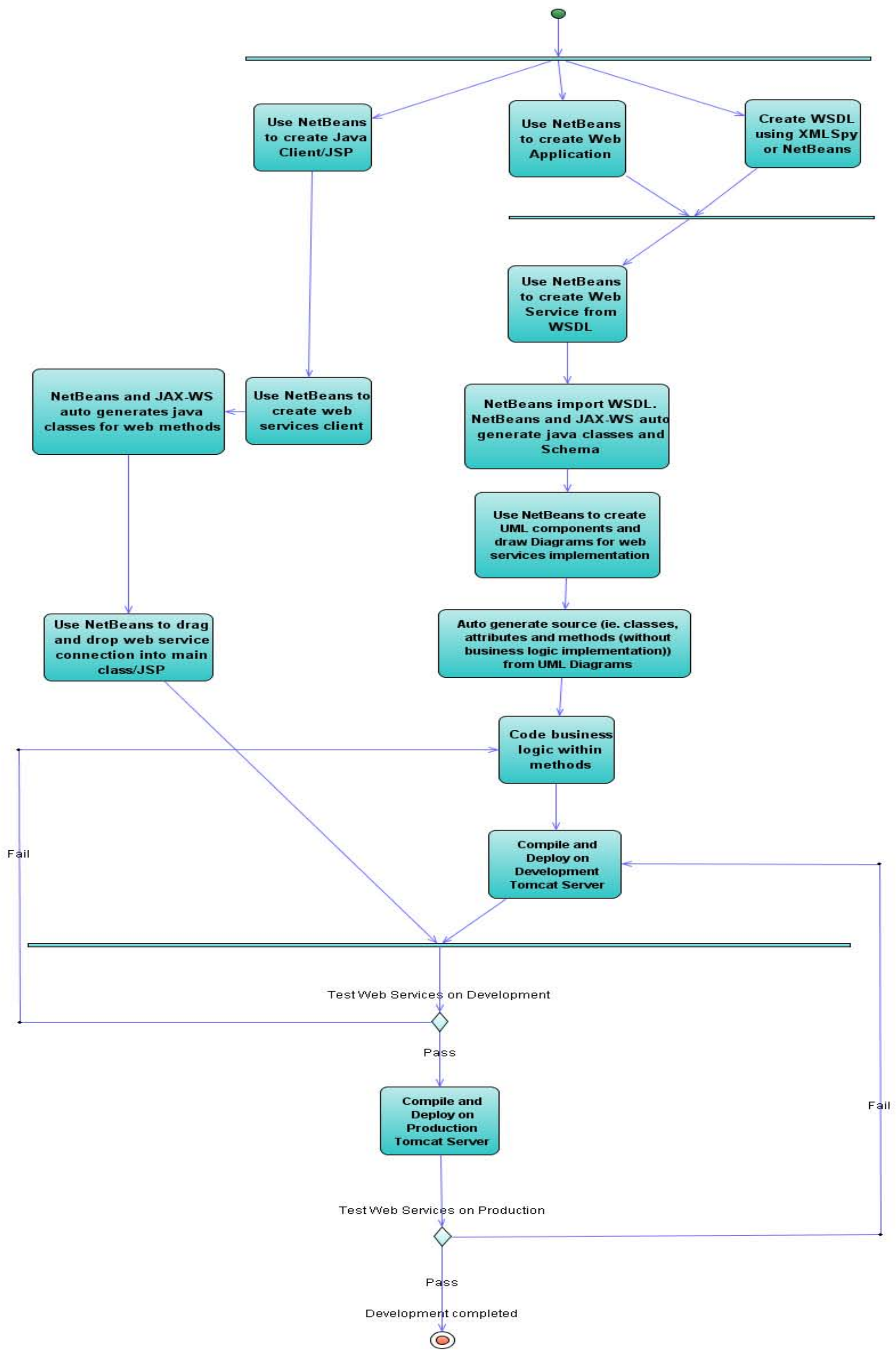

Use NetBeans to create Java Client/JSP
Use NetBeans to create Web Application
Create WSDL using XMLSpy or NetBeans
Use NetBeans to create Web Service from WSDL
NetBeans and JAX-WS auto generates java classes for web methods
Use NetBeans to create web services client
NetBeans import WSDL. NetBeans and JAX-WS auto generate java classes and Schema
Use NetBeans to create UML components and draw Diagrams for web services implementation
Use NetBeans to drag and drop web service connection into main class/JSP
Auto generate source (ie. classes, attributes and methods (without business logic implementation)) from UML Diagrams
Code business logic within methods
Fail
Compile and Deploy on Development Tomcat Server
Test Web Services on Development
Pass
Compile and Deploy on Production Tomcat Server
Fail
Test Web Services on Production
Pass
Development completed

Figure 24. Forward engineering using NetBeans, with WSDL configured separately and imported into NetBeans.

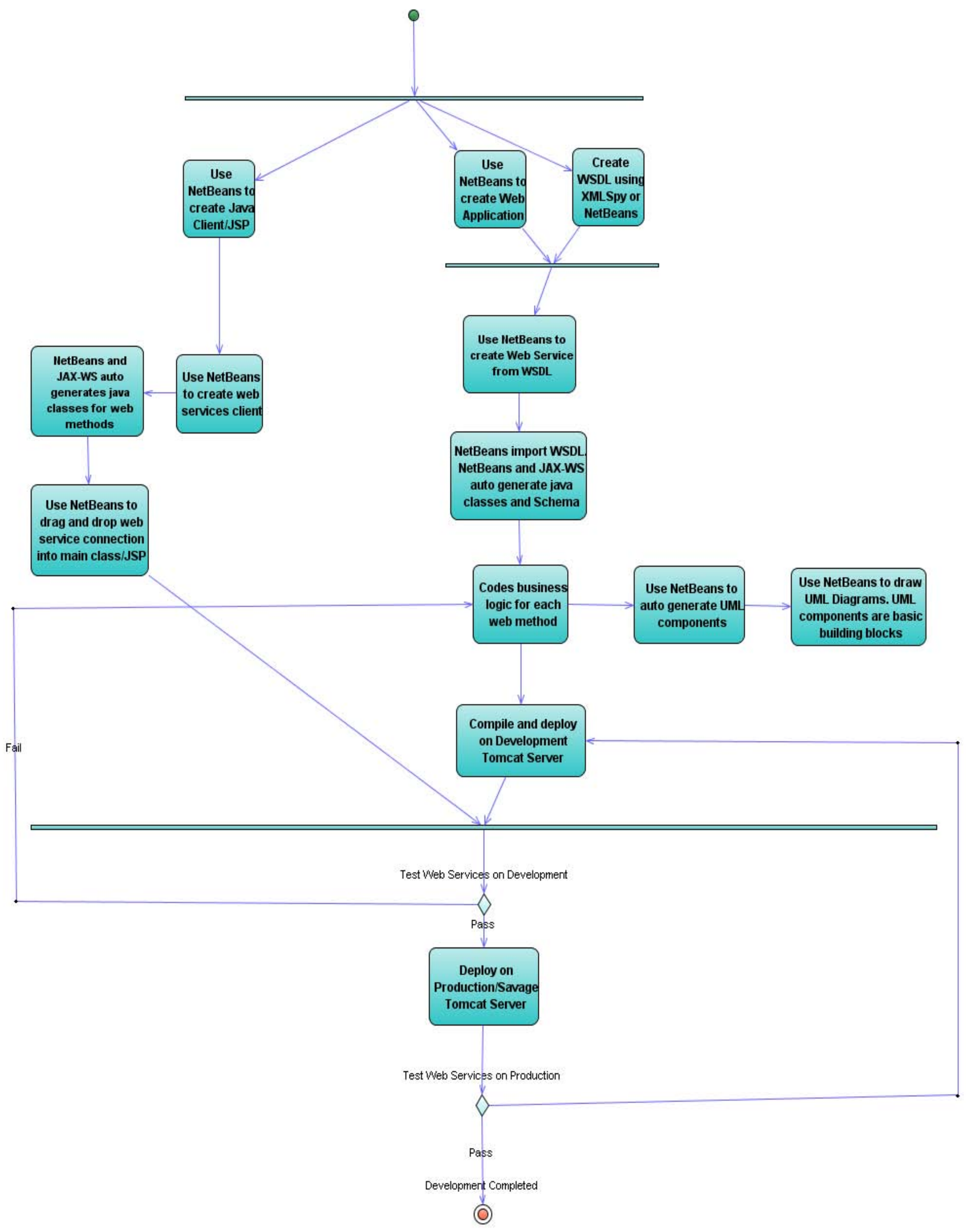

Use NetBeans to create Java Client/JSP
Use NetBeans to create Web Application
Create WSDL using XMLSpy or NetBeans
Use NetBeans to create Web Service from WSDL
NetBeans and JAX-WS auto generates java classes for web methods
Use NetBeans to create web services client
NetBeans import WSDL. NetBeans and JAX-WS auto generate java classes and Schema
Use NetBeans to drag and drop web service connection into main class/JSP
Codes business logic for each web method
Use NetBeans to auto generate UML components
Use NetBeans to draw UML Diagrams. UML components are basic building blocks
Fail
Compile and deploy on Development Tomcat Server
Test Web Services on Development
Pass
Deploy on Production/Savage Tomcat Server
Test Web Services on Production
Pass
Development Completed

Figure 25. Reverse engineering using NetBeans, with WSDL configured separately and imported into NetBeans.

The development process used for the development of SAVAGE web services is reverse engineering using NetBeans. JAX-WS auto-generates WSDL. The reason why this approach is chosen is because Java source code is developed first. The NetBeans UML models are reconstructed from the source code.

To create a project using NetBeans UML, use the NetBeans create project wizard. Select category UML and then Java-Platform Model. NetBeans will create a UML model in Java.

E. DESIGN COMPONENTS

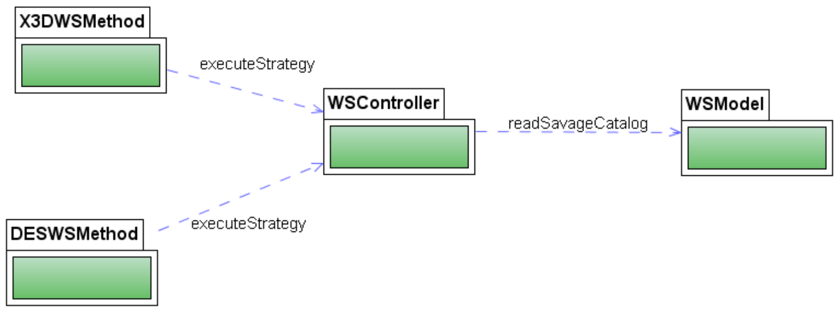


Figure 26. SAVAGE web services component diagram in UML. The components are the realization of the use cases.

There are four components in the demonstration design, as shown in Figure 26. Each component has its distinct function. The components are the realization of the use cases and design considerations. Each component is a Java package. The *X3DWSMethod* and *DESWSMethod* packages contain web methods for X3D and DES web services, respectively. The *WSController* package contains classes that implement the various business rules and logic for the web methods. The *WSModel* package implements the various entities or models. These entities are the basis for the business rules in *WSController* to work on. Class methods in *X3DWSMethod* and *DESWSMethod* do not use the class methods in *WSModel* directly. Instead, *WSController* plays the role of mediator between the various web methods and the common entity classes. This design approach facilitates component reuse such as *WSModel* being a common entity used by different web services.

F. DETAILED DESIGN

1. UML Class Diagram

Figure 27 shows the detailed design of classes in the *WSController* package.

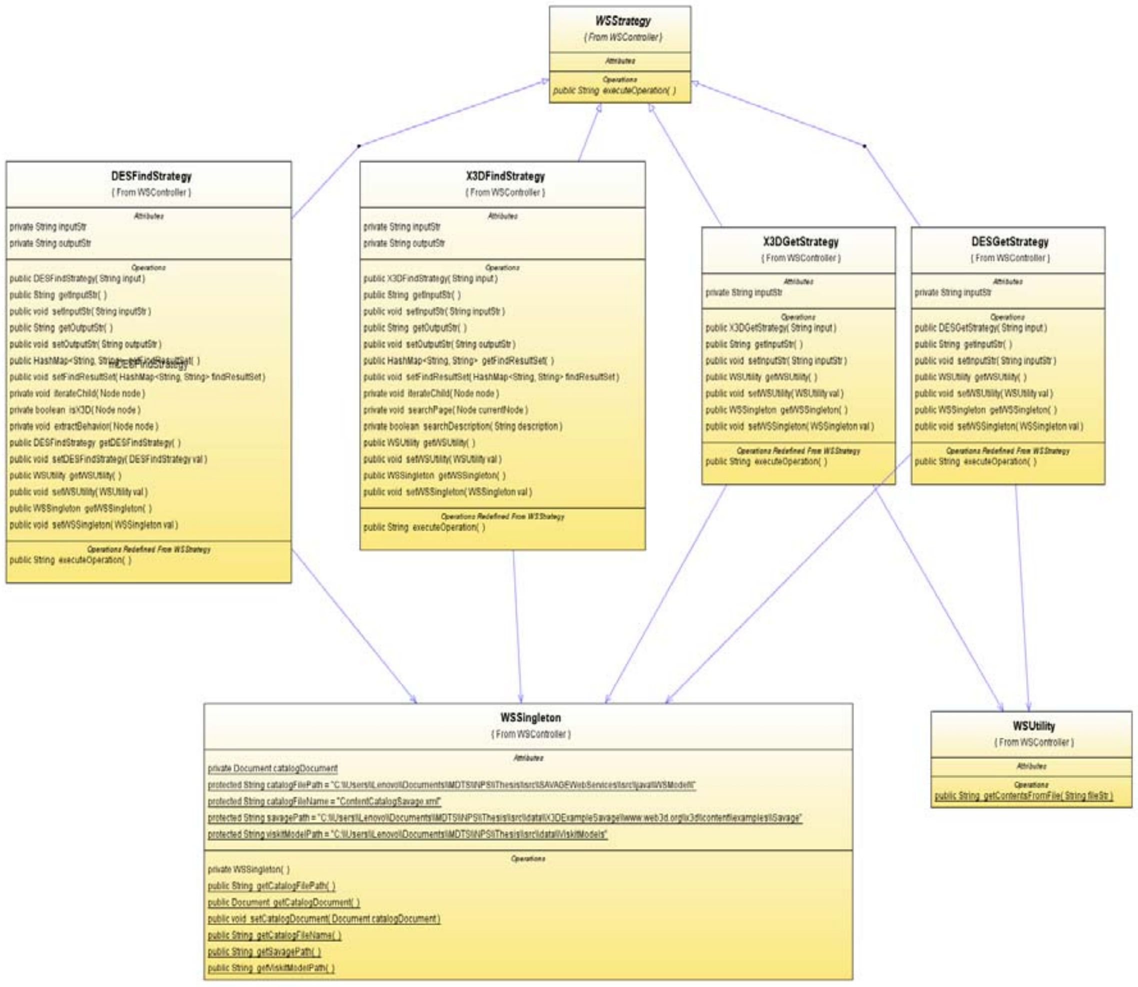

WSStrategy
(From WSController)
Attributes
Operations
public String executeOperation( )
DESFindStrategy
(From WSController)
Attributes
private String inputStr
private String outputStr
Operations
public DESFindStrategy(String input)
public String getInputStr( )
public void setInputStr(String inputStr)
public String getOutputStr( )
public void setOutputStr(String outputStr)
public HashMap<String, String> getFindResultSet( )
mDESFindStrategy
public void setFindResultSet(HashMap<String, String> findResultSet)
private void iterateChild(Node node)
private boolean isX3D(Node node)
private void extractBehavior(Node node)
public DESFindStrategy getDESFindStrategy( )
public void setDESFindStrategy(DESFindStrategy val)
public WSUtility getWSUtility( )
public void setWSUtility(WSUtility val)
public WSSingleton getWSSingleton( )
public void setWSSingleton(WSSingleton val)
Operations Redefined From WSStrategy
public String executeOperation( )
X3DFindStrategy
(From WSController)
Attributes
private String inputStr
private String outputStr
Operations
public X3DFindStrategy(String input)
public String getInputStr( )
public void setInputStr(String inputStr)
public String getOutputStr( )
public void setOutputStr(String outputStr)
public HashMap<String, String> getFindResultSet( )
public void setFindResultSet(HashMap<String, String> findResultSet)
private void iterateChild(Node node)
private void searchPage(Node currentNode)
private boolean searchDescription(String description)
public WSUtility getWSUtility( )
public void setWSUtility(WSUtility val)
public WSSingleton getWSSingleton( )
public void setWSSingleton(WSSingleton val)
Operations Redefined From WSStrategy
public String executeOperation( )
X3DGetStrategy
(From WSController)
Attributes
private String inputStr
Operations
public X3DGetStrategy(String input)
public String getInputStr( )
public void setInputStr(String inputStr)
public WSUtility getWSUtility( )
public void setWSUtility(WSUtility val)
public WSSingleton getWSSingleton( )
public void setWSSingleton(WSSingleton val)
Operations Redefined From WSStrategy
public String executeOperation( )
DESGetStrategy
(From WSController)
Attributes
private String inputStr
Operations
public DESGetStrategy(String input)
public String getInputStr( )
public void setInputStr(String inputStr)
public WSUtility getWSUtility( )
public void setWSUtility(WSUtility val)
public WSSingleton getWSSingleton( )
public void setWSSingleton(WSSingleton val)
Operations Redefined From WSStrategy
public String executeOperation( )
WSSingleton
(From WSController)
Attributes
private Document catalogDocument
protected String catalogFilePath = "C:\\Users\\Lenovo\\Documents\\MDTS\\NPS\\Thesis\\src\\SAVAGEWebServices\\src\\java\\WSModel\\"
protected String catalogFileName = "ContentCatalogSavage.xml"
protected String savagePath = "C:\\Users\\Lenovo\\Documents\\MDTS\\NPS\\Thesis\\src\\data\\X3DExampleSavage\\www.web3d.org\\x3d\\content\\examples\\Savage"
protected String viskitModelPath = "C:\\Users\\Lenovo\\Documents\\MDTS\\NPS\\Thesis\\src\\data\\ViskitModels"
Operations
private WSSingleton( )
public String getCatalogFilePath( )
public Document getCatalogDocument( )
public void setCatalogDocument(Document catalogDocument)
public String getCatalogFileName( )
public String getSavagePath( )
public String getViskitModelPath( )
WSUtility
(From WSController)
Attributes
Operations
public String getContentsFromFile(String fileStr)

Figure 27. SAVAGE web services classes implemented in *WSController* component. Class diagram in UML.

The Strategy pattern is a type of behavioral pattern. It comprises a set of classes, where each class implements a particular behavior. These implemented behaviors can be flexibly applied to the model, changing the way the application behaves on the fly. For SAVAGE web services implementation, the Strategy pattern is used to implement the business logic of each web method. Thus, there is one implemented strategy class corresponding to each web method. They are *X3DFindStrategy*, *X3DGetStrategy*, *DESFindStrategy* and *DESGetStrategy*. *WSStrategy* is the abstract class that implements abstract method *executeOperation*. The specific implementation of *executeOperation* is defined in the extended strategy classes. The context classes are *X3DWebService* and *DESWebService* in *X3DWSMethod* and *DESWSMethod* accordingly. The context classes use the strategies by invoking the abstract method *executeOperation*. *executeOperation* returns a string and the output (X3D model, DES model, etc) will depend on the implemented *executeOperation* defined in the extended strategy classes.

The extended strategy classes use *WSSingleton*. Singleton is a type of creational pattern. The singleton class creates and maintains the global static/single instance of class. Other classes use the singleton class to retrieve the global instance. For Savage web services, the static classes instances maintained by Singleton are *catalogDocument* (to constructed the Document Object Model (DOM) model for the SAVAGE catalog XML file), *catalogFilePath* (specific file directory where the SAVAGE catalog file is stored), *catalogFileName* (file name for the SAVAGE catalog file), *savageFilePath* (specific file directory for the root directory of the SAVAGE X3D repository; the precise location/subdirectory of a specific X3D model resides in its associated X3D URL) and *visualModelPath* (specific file directory for the root directory for the SAVAGE DES repository; the subdirectory resides in DES URL).

WSUtility basically contains reusable utility operations such as *getContentFromFile*.

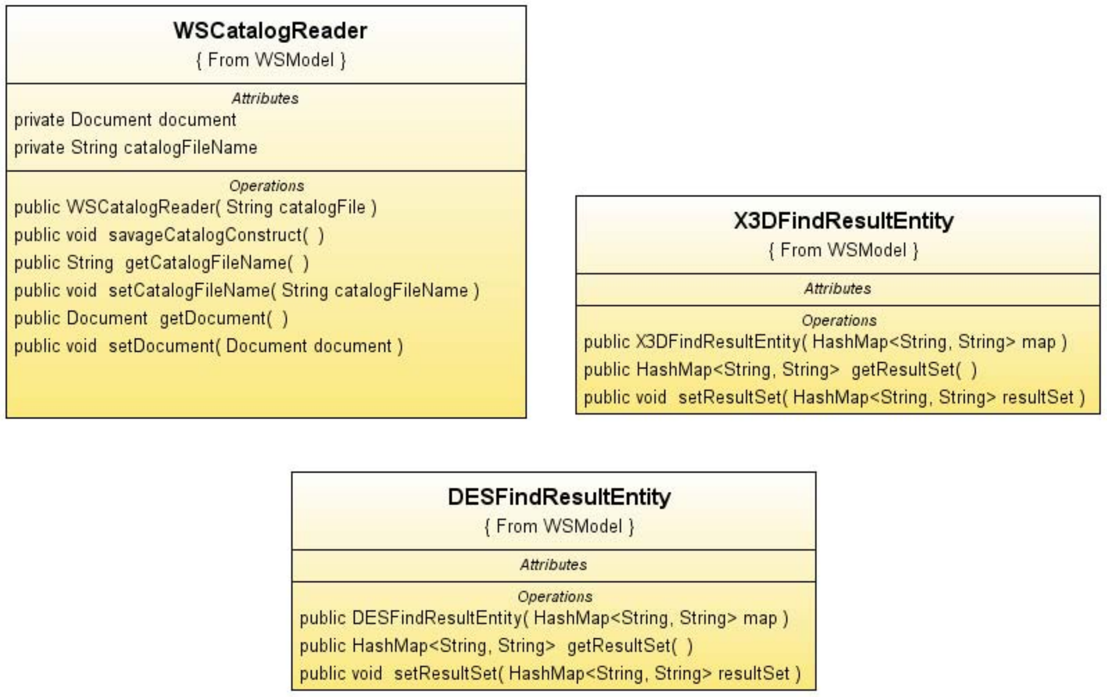
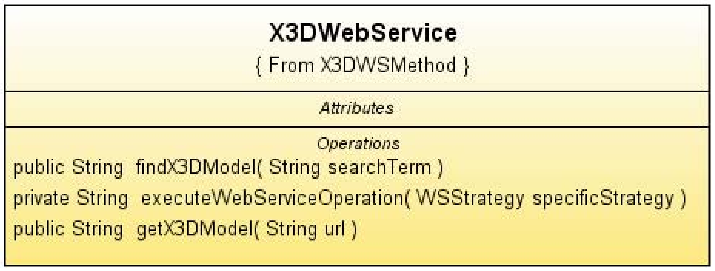

WSCatalogReader
{ From WSModel }
Attributes
private Document document
private String catalogFileName
Operations
public WSCatalogReader(String catalogFile)
public void savageCatalogConstruct( )
public String getCatalogFileName( )
public void setCatalogFileName(String catalogFileName)
public Document getDocument( )
public void setDocument(Document document)
X3DFindResultEntity
{ From WSModel }
Attributes
Operations
public X3DFindResultEntity(HashMap<String, String> map)
public HashMap<String, String> getResultSet( )
public void setResultSet(HashMap<String, String> resultSet)
DESFindResultEntity
{ From WSModel }
Attributes
Operations
public DESFindResultEntity(HashMap<String, String> map)
public HashMap<String, String> getResultSet( )
public void setResultSet(HashMap<String, String> resultSet)

Figure 28. SAVAGE web services classes implemented in the *WSModel* component. Class diagram in UML.

There are three classes in the *WSModel* package (Figure 28). They are *WSCatalogReader*, *X3DFindResultEntity* and *DESFindResultEntity*. *WSCatalogReader* reads in and builds DOM for the SAVAGE catalog file. *X3DFindResultEntity* class encapsulates the result (list of X3D URLs and names) computed by *X3DFindStrategy* in HashMap. *X3DFindResultEntity* will be parsed into XML format before returning to the *findX3DModel* web method. Likewise, *DESFindResultEntity* class is used to encapsulate the result (list of DES agents and URLs) from *DESFindStrategy*.


X3DWebService
{ From X3DWSMethod }
Attributes
Operations
public String findX3DModel(String searchTerm)
private String executeWebServiceOperation(WSStrategy specificStrategy)
public String getX3DModel(String url)

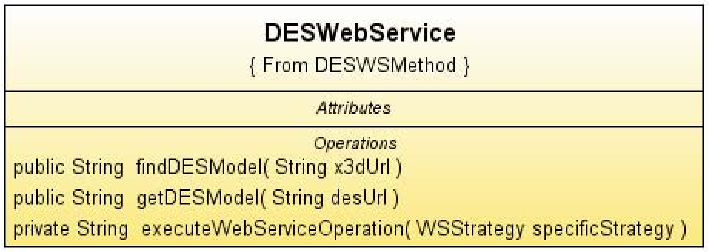

DESWebService
{ From DESWSMethod }
Attributes
Operations
public String findDESModel(String x3dUrl)
public String getDESModel(String desUrl)
private String executeWebServiceOperation(WSStrategy specificStrategy)

Figure 29. SAVAGE web services classes implemented in *X3DWSMethod and DESWSMethod* components. Class diagram in UML.

X3DWebService implements web method *findX3DModel* which takes in *searchTerm* as parameter and returns a list of X3D URLs and names in XML. *getX3DModel* is the implemented web method to retrieve an X3D model given X3D URL. *DESWebService* implements web method *getDESModel* and *findDESModel*. *ExecuteWebServiceOperation* is a generic private method and its parameter is the abstract class *WSStrategy*. In other words, it can be used by the different web methods. Its role is to facilitate the execution of specific strategy passed in by the web methods.

2. UML Sequence Diagram

The UML sequence diagram in Figure 30 shows the sequence of interactions among different class instances. The detailed interactions between classes can be complicated, hence it is important to capture and focus on the essential class interactions.

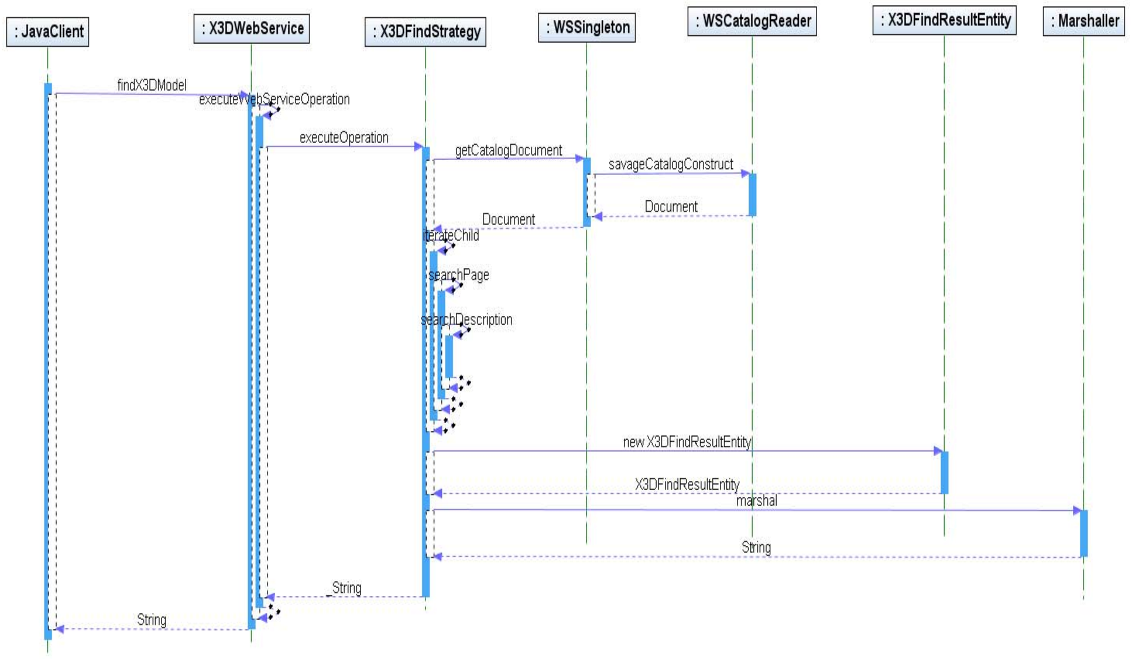


Figure 30. SAVAGE web services sequence diagram in UML for *findX3DModel* web method.

JavaClient invokes *findX3DModel* web method. *findX3DModel* performs *executeWebServiceOperation*, which then invokes *executeOperation*. *X3DFindStrategy getCatalogDocument* from *WSSingleton*. *WSSingleton* executes *savageCatalogConstruct* if the DOM for the SAVAGE catalog is not constructed. *X3DFindStrategy* will then iterate through the nodes in DOM. When it encounters page construct, it performs *searchPage* which does a string comparison between *searchTerm* and the contents of name and title attributes. *searchPage* also invokes *searchDescription*. *SearchDescription* tokenizes the description into keywords. Each keyword is then compared to *searchTerm* to identify matches. *X3DFindEntity* is constructed to store the list of relevant X3D URLs and names. *X3DFindStrategy* then marshala *X3DFindEntity* into an XML String via *Marshaller*. This XML String is returned to *X3DWebService* before it is returned to *JavaClient* via the *findX3DModel* web method.

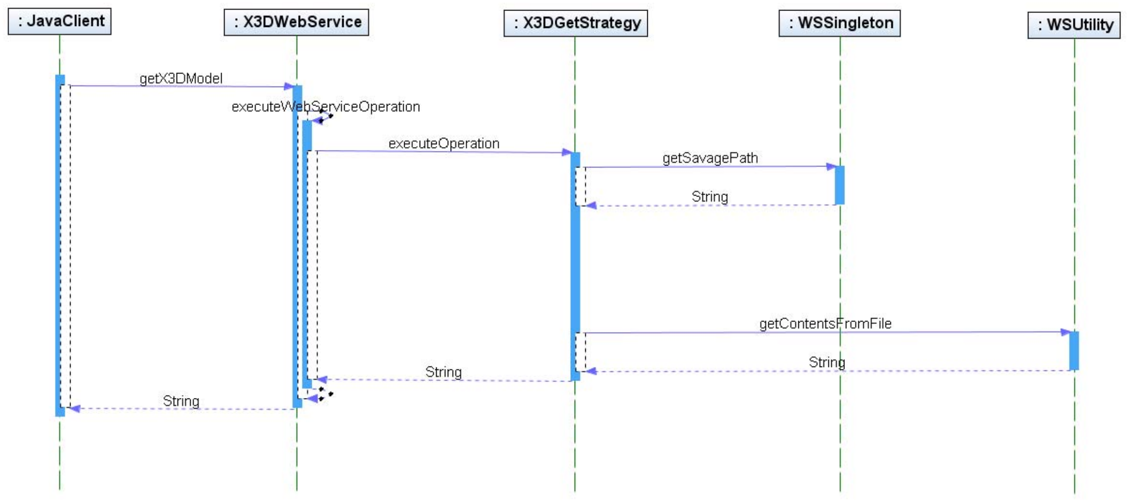


Figure 31. SAVAGE web services sequence diagram in UML for *getX3DModel* web method.

Figure 31 shows the UML sequence diagram for the *getX3DModel* web method. *JavaClient* invokes *getX3DModel* web method with X3D URL as the input parameter. After the invocation of *executeOperation*, *X3DGetStrategy* invokes *getSavagePath* from *WSSingleton*. *getSavagePath* returns the root directory of the SAVAGE X3D repository. The specific location and name of the X3D model can be extracted from the X3D URL. *X3DGetStrategy* then calls *getContentsFromFile* and the X3D model is returned to *JavaClient* via *X3DWebService*.

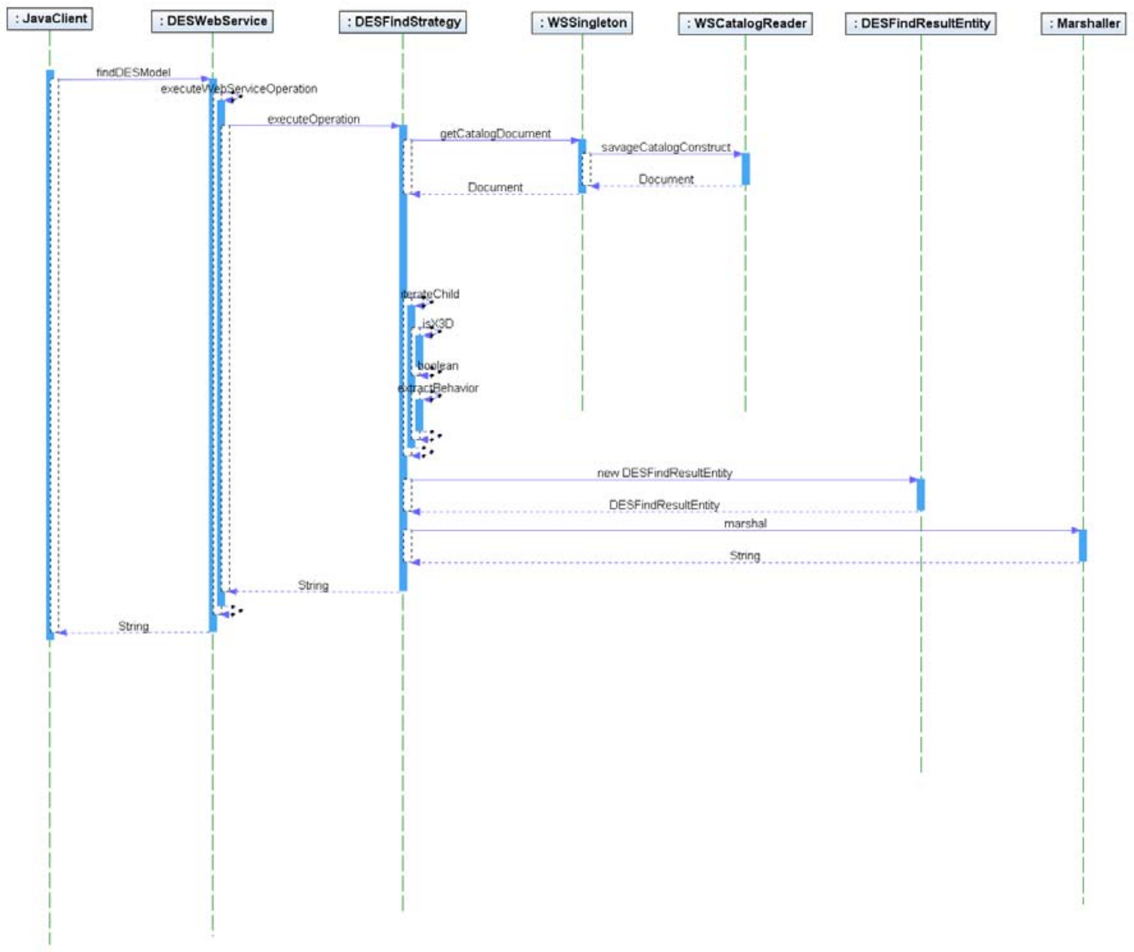


Figure 32. SAVAGE web services sequence diagram in UML for *findDESModel* web method.

JavaClient invokes the *findDESModel* web method with X3D URL as the input parameter (Figure 32). Upon execution of *executeOperation*, *DESFindStrategy* invokes *getCatalogDocument* (SAVAGE catalog in DOM) from *WSSingleton*. *DESFindStrategy* iterates across the DOM. For each page element, *DESFindStrategy* performs the *isX3D* check. *isX3D* does a string comparison between the input parameter of the *findDESModel* web method and the URL attribute of the page construct. If the X3D URL matches, it calls *extractBehavior* from the *BehaviorParameterSet* element. *extractBehavior* reads the content from agent and URL attributes from the *SimulationAgent* element. *DESFindStrategy* instantiates *DESFindResultEntity* to store the list of agent and DES URLs in a class attribute of type HashMap. *DESFindResultEntity* is marshaled into XML before returning to *JavaClient* via the *findDESModel* web method.

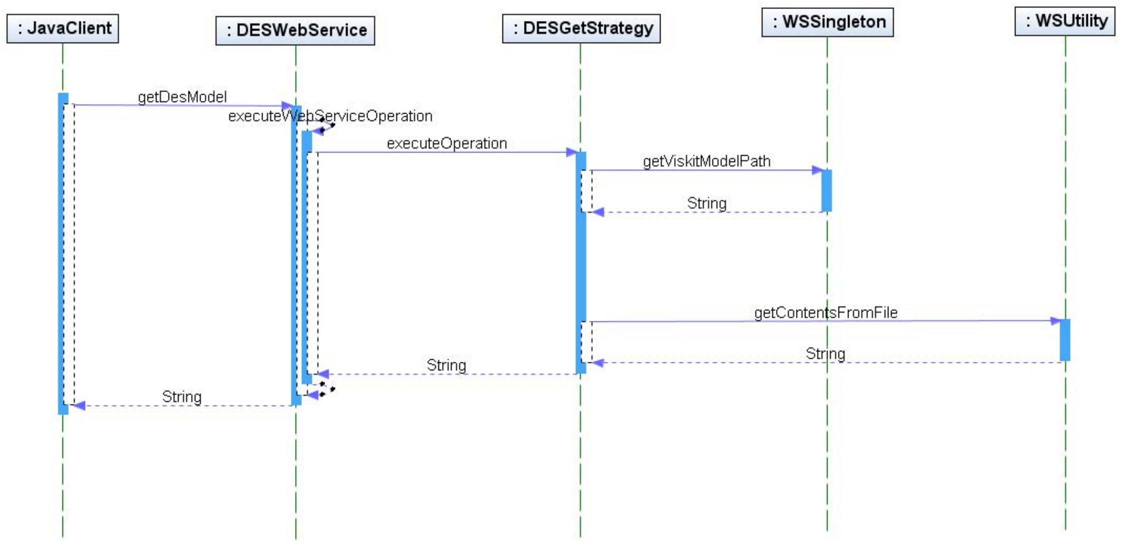


Figure 33. SAVAGE web services sequence diagram in UML for *getDESModel* web method.

JavaClient invokes the *getDESModel* web method with DES URL as the input parameter (Figure 33). Upon *executeOperation*, *DESGetStrategy* triggers *getViskitModelPath*, which returns the Viskit model root directory. The specific sub-directory and the name of the DES model file can be extracted from DES URL. *DESGetStrategy* calls *WSUtility* method *getContentFromFile* to read the DES model file from the SAVAGE Viskit repository. The DES model is then returned to *JavaClient* via the *getDesModel* web method.

3. SAVAGE WSDL

The Java API for XML Web Services (JAX-WS) generates WSDL files for SAVAGE web services. The implemented Java web method is in class *X3DWebService*.

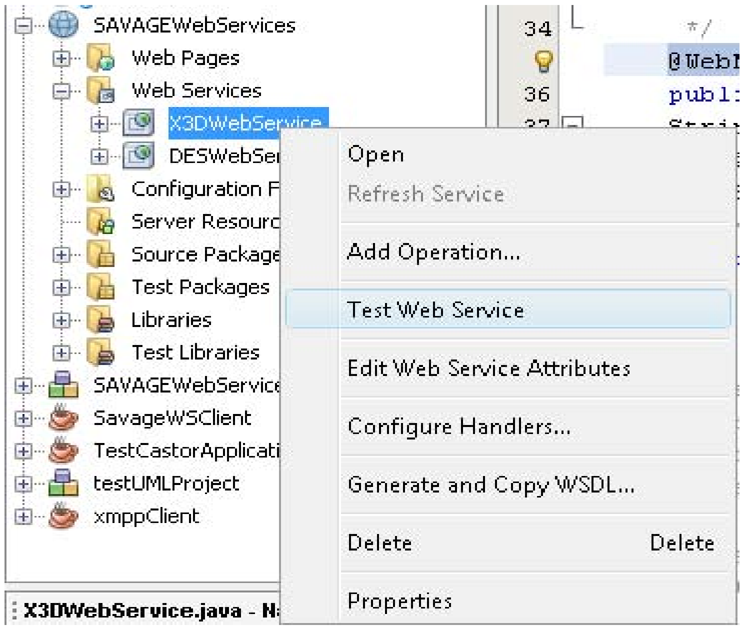
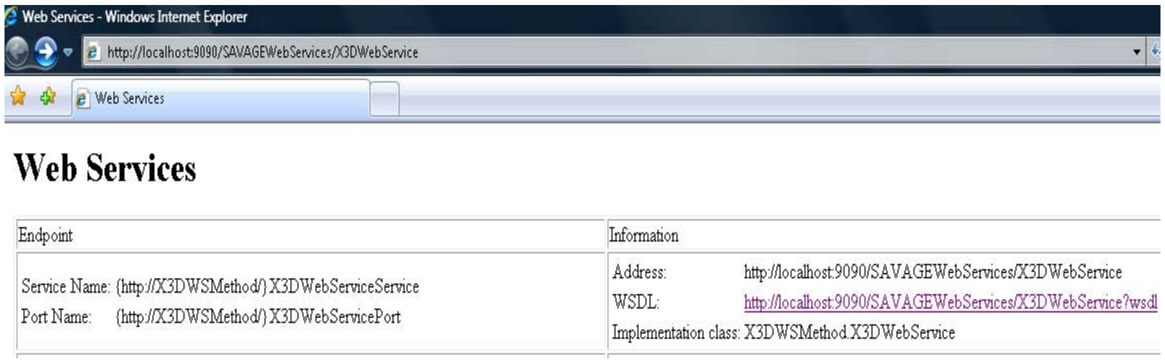


Figure 34. Test web service using NetBeans.

To view the WSDL file, simply right click "*X3DWebService*" under the Web Services source directory and select "Test Web Service" (see Figure 34).



Web Services - Windows Internet Explorer

http://localhost:9090/SAVAGEWebServices/X3DWebService

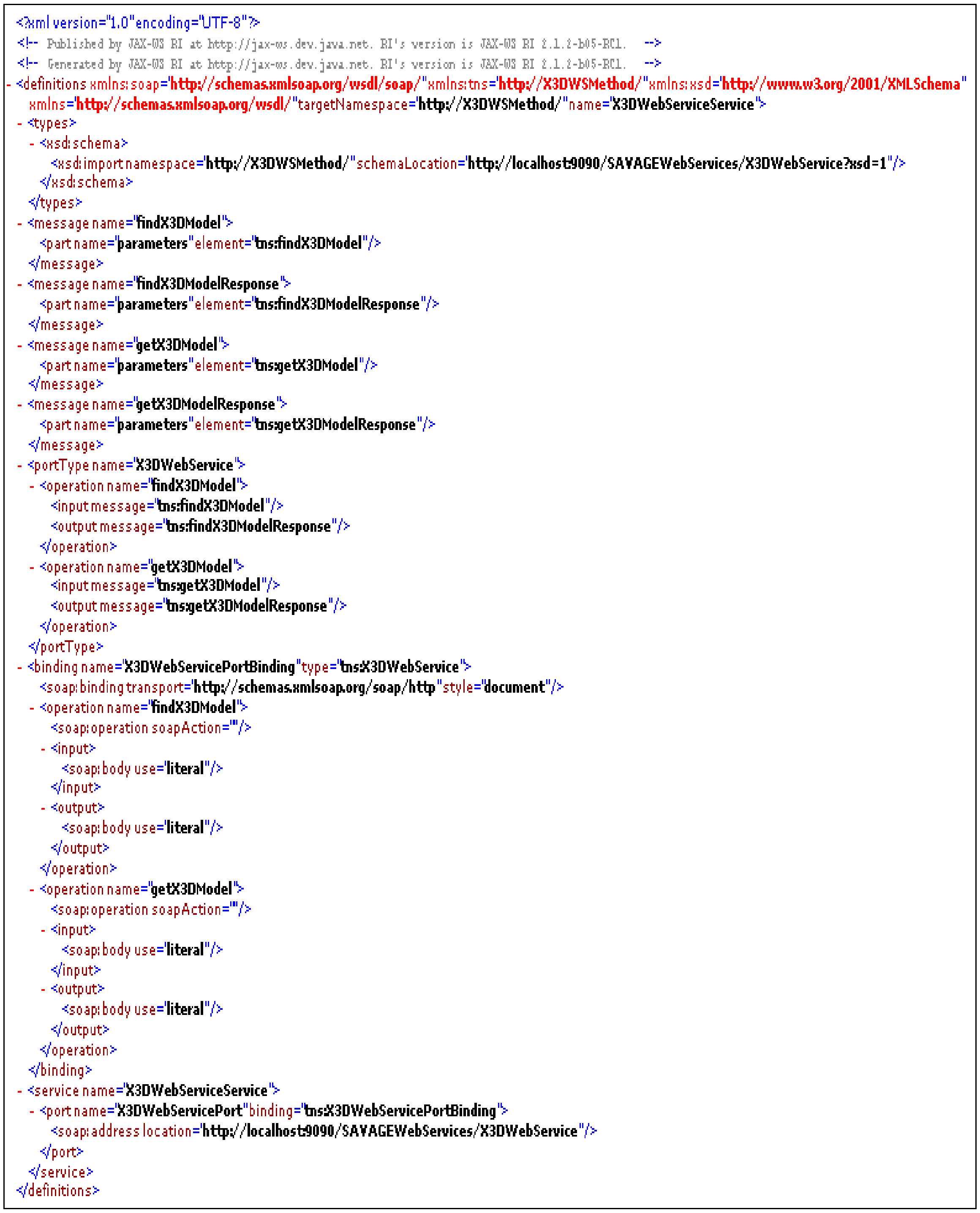
Web Services

Web Services

Endpoint	Information
Service Name: {http://X3DWSMethod/}X3DWebServiceService Port Name: {http://X3DWSMethod/}X3DWebServicePort	Address: http://localhost:9090/SAVAGEWebServices/X3DWebService WSDL: http://localhost:9090/SAVAGEWebServices/X3DWebService?wsdl Implementation class: X3DWSMethod.X3DWebService

Figure 35. Web browser displays hyperlink to the *X3DWebService* WSDL file.

The endpoint of the web service will be displayed on the web browser. The address of *X3DWebService* in the development environment is http://localhost:9090/SAVAGEWebServices/X3DWebService (Figure 35). The WSDL resides in http://localhost:9090/SAVAGEWebServices/X3DWebService?wsdl. The link to the WSDL is clicked. The generated WSDL file for the X3D web service is as shown below (Figure 36). The server is *localhost* because the current development environment is the author's personal laptop. If web services are hosted on another web application server, the specific name and port number of the server is used instead.

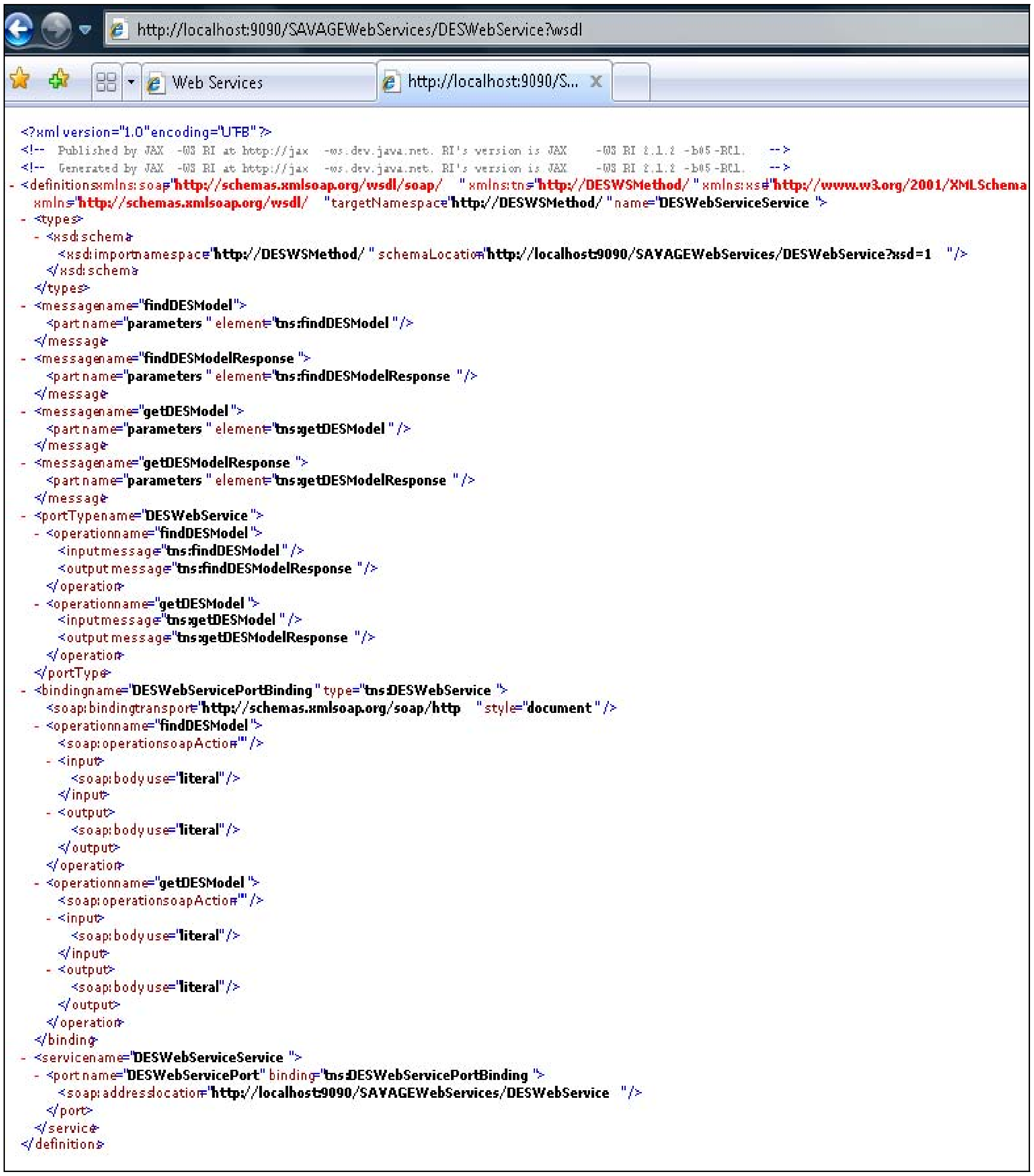


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.2-b05-RC1. -->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.2-b05-RC1. -->
- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://X3DWSMethod/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://X3DWSMethod/" name="X3DWebServiceService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://X3DWSMethod/" schemaLocation="http://localhost:9090/SAVAGEWebServices/X3DWebService?xsd=1"/>
      </xsd:schema>
    </types>
  - <message name="findX3DModel">
      <part name="parameters" element="tns:findX3DModel"/>
    </message>
  - <message name="findX3DModelResponse">
      <part name="parameters" element="tns:findX3DModelResponse"/>
    </message>
  - <message name="getX3DModel">
      <part name="parameters" element="tns:getX3DModel"/>
    </message>
  - <message name="getX3DModelResponse">
      <part name="parameters" element="tns:getX3DModelResponse"/>
    </message>
  - <portType name="X3DWebService">
    - <operation name="findX3DModel">
        <input message="tns:findX3DModel"/>
        <output message="tns:findX3DModelResponse"/>
      </operation>
    - <operation name="getX3DModel">
        <input message="tns:getX3DModel"/>
        <output message="tns:getX3DModelResponse"/>
      </operation>
    </portType>
  - <binding name="X3DWebServicePortBinding" type="tns:X3DWebService">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="findX3DModel">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    - <operation name="getX3DModel">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
  - <service name="X3DWebServiceService">
    - <port name="X3DWebServicePort" binding="tns:X3DWebServicePortBinding">
        <soap:address location="http://localhost:9090/SAVAGEWebServices/X3DWebService"/>
      </port>
    </service>
</definitions>
```

Figure 36. WSDL file that describes *X3DWebService*.

The WSDL describes information about *X3DWebService* which includes the web services operations (*findX3DModel* and *getX3DModel*), SOAP binding and the web services endpoint information such as web service address location.

The address of *DESWebService* on development environment is http://localhost:9090/SAVAGEWebServices/DESWebService. The WSDL resides in http://localhost:9090/SAVAGEWebServices/DESWebService?wsdl. The generated WSDL for *DESWebService* is as shown below (Figure 37).



http://localhost:9090/SAVAGEWebServices/DESWebService?wsdl
Web Services
http://localhost:9090/S...

```
<?xml version="1.0" encoding="UTF8" ?>
<!-- Published by JAX -WS RI at http://jax -ws.dev.java.net. RI's version is JAX -WS RI 2.1.2 -b05 -RC1. -->
<!-- Generated by JAX -WS RI at http://jax -ws.dev.java.net. RI's version is JAX -WS RI 2.1.2 -b05 -RC1. -->
- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://DESWSMethod/" xmlns:xsd="http://www.w3.org/2001/XMLSchema
   xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://DESWSMethod/" name="DESWebServiceService">
 - <types>
  - <xsd:schema>
      <xsd:import namespace="http://DESWSMethod/" schemaLocation="http://localhost:9090/SAVAGEWebServices/DESWebService?xsd=1" />
    </xsd:schema>
   </types>
 - <message name="findDESModel">
    <part name="parameters" element="tns:findDESModel" />
   </message>
 - <message name="findDESModelResponse">
    <part name="parameters" element="tns:findDESModelResponse" />
   </message>
 - <message name="getDESModel">
    <part name="parameters" element="tns:getDESModel" />
   </message>
 - <message name="getDESModelResponse">
    <part name="parameters" element="tns:getDESModelResponse" />
   </message>
 - <portType name="DESWebService">
  - <operation name="findDESModel">
      <input message="tns:findDESModel" />
      <output message="tns:findDESModelResponse" />
    </operation>
  - <operation name="getDESModel">
      <input message="tns:getDESModel" />
      <output message="tns:getDESModelResponse" />
    </operation>
   </portType>
 - <binding name="DESWebServicePortBinding" type="tns:DESWebService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  - <operation name="findDESModel">
      <soap:operation soapAction="" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  - <operation name="getDESModel">
      <soap:operation soapAction="" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
   </binding>
 - <service name="DESWebServiceService">
  - <port name="DESWebServicePort" binding="tns:DESWebServicePortBinding">
      <soap:address location="http://localhost:9090/SAVAGEWebServices/DESWebService" />
    </port>
   </service>
</definitions>
```

Figure 37. WSDL file that describes *DESWebService*.

G. RESOURCES AUTOGENERATED BY NETBEANS AND JAX-WS

NetBeans generates Java source which imports the JAX-WS library. These classes are *DESWebService.java* and *X3DWebService.java*. Web services are indicated with @WebService(). Web methods are indicated with @WebMethod. In the case of SAVAGE web services development, there are three types of resources auto-generated by JAX-WS. JAX-WS generates web method Java classes. Java classes generated for the X3D web service are *FindX3DModel.java*, *FindX3DModelResponse.java*, *GetX3DModel.java* and *GetX3DModelResponse.java*. Java classes generated for the DES web service are *FindDESModel.java*, *FindDESModelResponse.java*, *GetDESModel.java* and *GetDESModelResponse.java*. Basically, these are the classes required to generate the other two resources, which are the WSDL and the associated schemas for X3D and DES web services. Java Architecture for XML Bindings (JAXB) marshals the Java classes to XML.

H. EXTENDING SAVAGE WEB SERVICES

Additional web methods for SAVAGE web services are needed. For example, *X3DWebService* might incorporate web methods that perform add, delete and update of X3D models. Hence, the SAVAGE web services design has to be extensible. In principle, there are three simple steps to extend SAVAGE web services.

Step 1 – If it is an added web method for an existing web service (e.g., *X3DWebService*), then add the class method into the existing web service class. If the required web method does not belong to the existing web service, then create a new *[model name]WSMethod* package, create a new web service class *[model name]WebService* in the package and insert the web method into the newly created class. Note that web method calls generic private method *executeWebServiceOperation* (with input parameter *WSStrategy*) which then performs standard invocation of the specific strategy passed in by the web method.

Step 2 – Changes are required in the *WSModel* package. The reusable SAVAGE catalog model does not meet the design requirement. The design approach is to add a builder class to construct the new model of interest. If the model is meant to be a global instance, then use *WSSingleton* to create and manage the model instance. Create new result entity classes if there is a design requirement to parse the result set into XML.

Step 3 – Changes are required to the *WSController* package. Extend *WSStrategy* for each web method added. Each extended strategy class *[model name][function]Strategy* will implement *executeOperation* which encapsulates the business logic or rules for the web method.

Enhancements can be performed in two ways:

1. Insert web methods, strategy, model classes and their associated methods in the NetBeans UML. Invoke NetBeans forward engineering to generate the necessary code stubs for the classes and methods. Proceed with detailed implementation within the created class and methods. Perform NetBeans reverse engineering to update the UML models after implementation. Test the web method implementation before checking into SVN.
2. Implement directly in the code base. Update UML model via NetBeans reverse engineering. Test the web method implementation before checking into SVN.

I. SUMMARY

This chapter presents use cases and various implementation approaches for SAVAGE web services. The components of SAVAGE web services were described in a UML component diagram. The class hierarchy for each component was addressed in UML class diagrams. Interactions between class instantiations were illustrated in UML sequence diagrams. The chapter concluded by documenting the design approach to extend SAVAGE web services.

THIS PAGE INTENTIONALLY LEFT BLANK

V. IMPLEMENTATION, DEPLOYMENT AND RESULTS

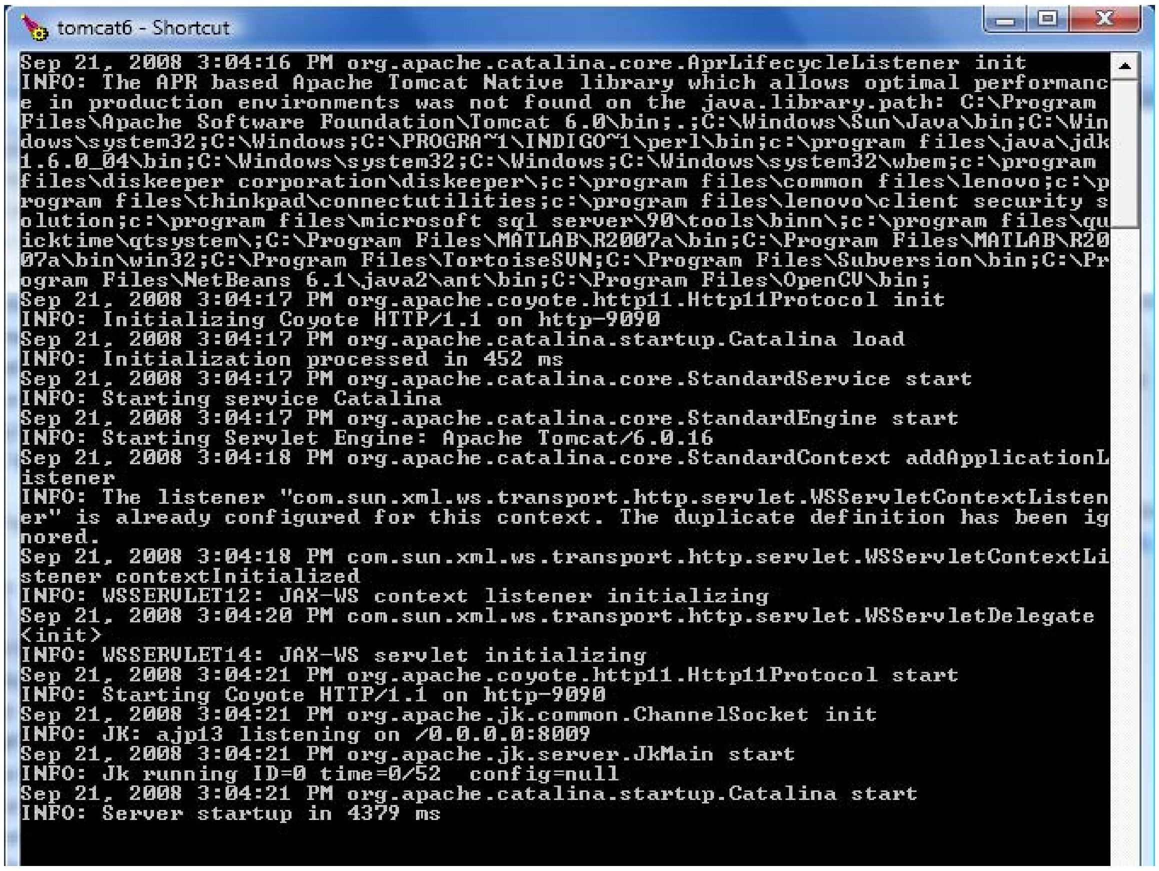
A. INTRODUCTION

This chapter starts off by elaborating the setup for the development environment. The hardware and software required for the development of SAVAGE web services are discussed. The implemented SAVAGE web services are deployed on SAVAGE servers. The chapter concludes by discussing test results corresponding to each use case.

B. IMPLEMENTATION SETUP

Taking into consideration the design specified in Chapter IV, the hardware required to support the demonstration on the development environment is simple. It is a single machine setup. The machine runs on a 32-bit operating system, Intel® Core™ 2 Duo CPU T7300 @ 2.00GHz, with 2 GB RAM and runs on VISTA. If the application runs on this machine and meets the design requirement, then it will likely run on other types of machine (e.g., Mac, Unix, etc) with equivalent specifications.

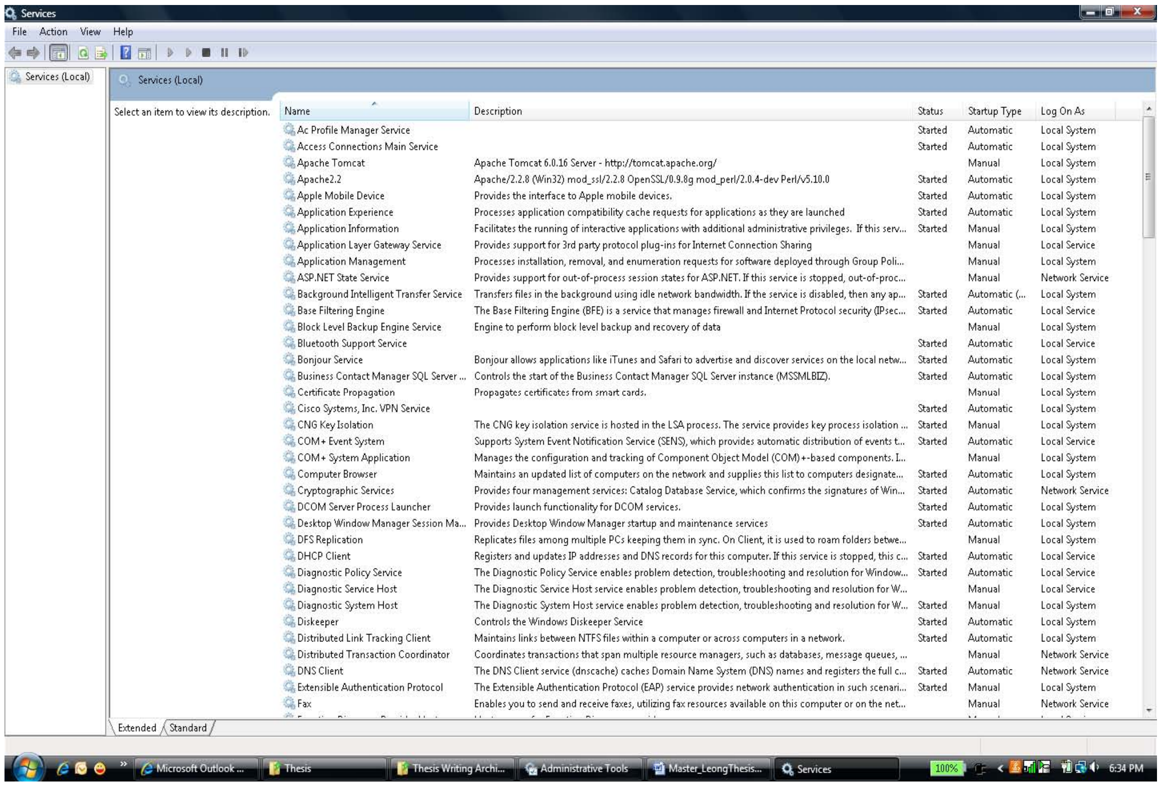
The software required to be installed on the development and demonstration machine is NetBeans Integrated Development Environment (IDE) 6.1 (version 6.0 will do as well), Java Development Kit (JDK) version 6 (version 5 will do, too) and Tomcat application server version 6. The NetBeans IDE is used for the implementation of web services and to facilitate web services deployment.



```
Sep 21, 2008 3:04:16 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The APR based Apache Tomcat Native library which allows optimal performanc
e in production environments was not found on the java.library.path: C:\Program
Files\Apache Software Foundation\Tomcat 6.0\bin;.;C:\Windows\Sun\Java\bin;C:\Win
dows\system32;C:\Windows;C:\PROGRA~1\INDIGO~1\perl\bin;c:\program files\java\jdk
1.6.0_04\bin;C:\Windows\system32;C:\Windows;C:\Windows\system32\wbem;c:\program
files\diskeeper corporation\diskeeper\;c:\program files\common files\lenovo;c:\p
rogram files\thinkpad\connectutilities;c:\program files\lenovo\client security s
olution;c:\program files\microsoft sql server\90\tools\binn\;c:\program files\qu
icktime\qtsystem\;C:\Program Files\MATLAB\R2007a\bin;C:\Program Files\MATLAB\R20
07a\bin\win32;C:\Program Files\TortoiseSVN;C:\Program Files\Subversion\bin;C:\Pr
ogram Files\NetBeans 6.1\java2\ant\bin;C:\Program Files\OpenCV\bin;
Sep 21, 2008 3:04:17 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-9090
Sep 21, 2008 3:04:17 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 452 ms
Sep 21, 2008 3:04:17 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Sep 21, 2008 3:04:17 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.16
Sep 21, 2008 3:04:18 PM org.apache.catalina.core.StandardContext addApplicationL
istener
INFO: The listener "com.sun.xml.ws.transport.http.servlet.WSServletContextListen
er" is already configured for this context. The duplicate definition has been ig
nored.
Sep 21, 2008 3:04:18 PM com.sun.xml.ws.transport.http.servlet.WSServletContextLi
stener contextInitialized
INFO: WSSERVLET12: JAX-WS context listener initializing
Sep 21, 2008 3:04:20 PM com.sun.xml.ws.transport.http.servlet.WSServletDelegate
<init>
INFO: WSSERVLET14: JAX-WS servlet initializing
Sep 21, 2008 3:04:21 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-9090
Sep 21, 2008 3:04:21 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Sep 21, 2008 3:04:21 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/52  config=null
Sep 21, 2008 3:04:21 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 4379 ms
```

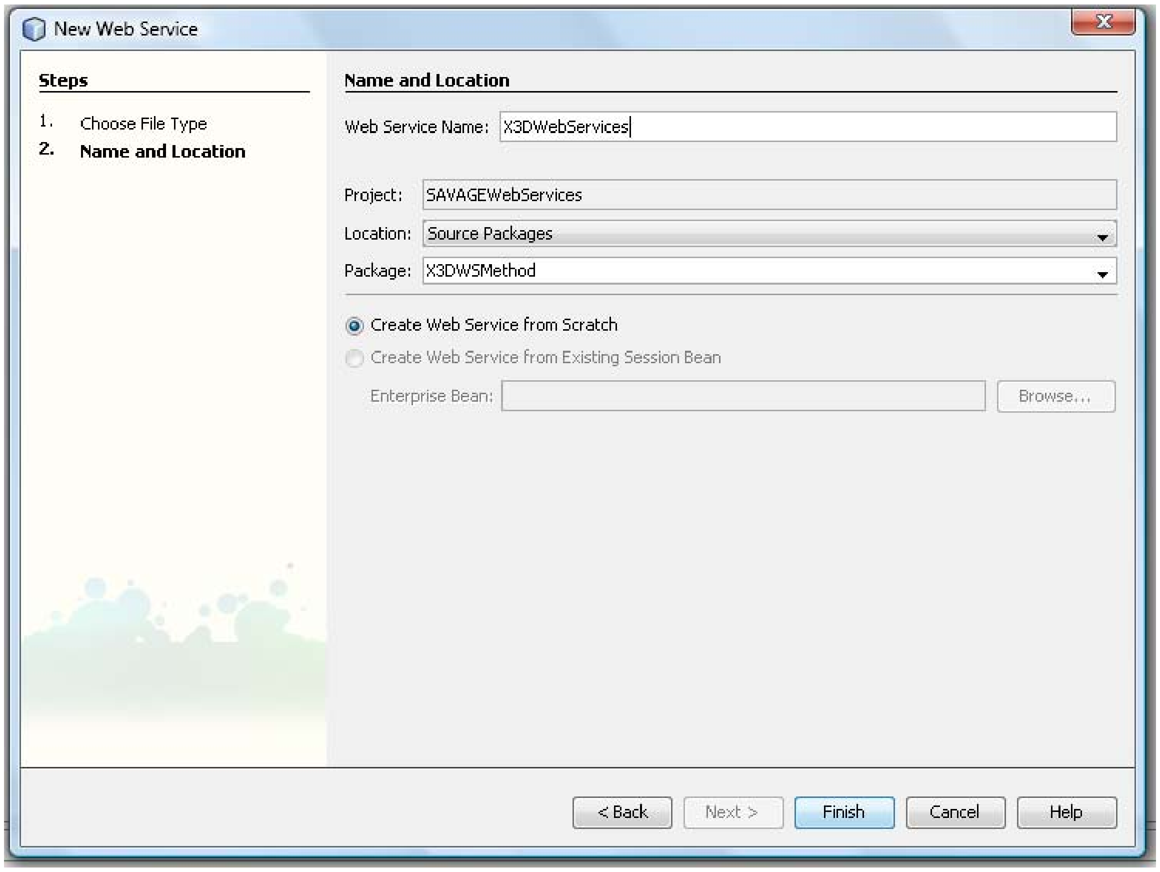
Figure 38. View Tomcat application server log in command prompt.

The Tomcat application server can run off window service or command prompt (run as administrator). Running the Tomcat application server in command prompt makes it more convenient to test and troubleshoot because you can view the server log running off the command prompt (Figure 38) while coding in NetBeans. The executable file for running Tomcat application can be configured as window service. To manage window services, go to Control Panel → Administrative Tools → Services. Select the service and click on the play or stop icons to start or terminate window service (See Figure 39).



Name	Description	Status	Startup Type	Log On As
Ac Profile Manager Service		Started	Automatic	Local System
Access Connections Main Service		Started	Automatic	Local System
Apache Tomcat	Apache Tomcat 6.0.16 Server - http://tomcat.apache.org/		Manual	Local System
Apache2.2	Apache/2.2.8 (Win32) mod_ssl/2.2.8 OpenSSL/0.9.8g mod_perl/2.0.4-dev Perl/v5.10.0	Started	Automatic	Local System
Apple Mobile Device	Provides the interface to Apple mobile devices.	Started	Automatic	Local System
Application Experience	Processes application compatibility cache requests for applications as they are launched	Started	Automatic	Local System
Application Information	Facilitates the running of interactive applications with additional administrative privileges. If this serv...	Started	Manual	Local System
Application Layer Gateway Service	Provides support for 3rd party protocol plug-ins for Internet Connection Sharing		Manual	Local Service
Application Management	Processes installation, removal, and enumeration requests for software deployed through Group Poli...		Manual	Local System
ASP.NET State Service	Provides support for out-of-process session states for ASP.NET. If this service is stopped, out-of-proc...		Manual	Network Service
Background Intelligent Transfer Service	Transfers files in the background using idle network bandwidth. If the service is disabled, then any ap...	Started	Automatic (...	Local System
Base Filtering Engine	The Base Filtering Engine (BFE) is a service that manages firewall and Internet Protocol security (IPsec...	Started	Automatic	Local Service
Block Level Backup Engine Service	Engine to perform block level backup and recovery of data		Manual	Local System
Bluetooth Support Service		Started	Automatic	Local Service
Bonjour Service	Bonjour allows applications like iTunes and Safari to advertise and discover services on the local netw...	Started	Automatic	Local System
Business Contact Manager SQL Server ...	Controls the start of the Business Contact Manager SQL Server instance (MSSMLBIZ).	Started	Automatic	Local System
Certificate Propagation	Propagates certificates from smart cards.		Manual	Local System
Cisco Systems, Inc. VPN Service		Started	Automatic	Local System
CNG Key Isolation	The CNG key isolation service is hosted in the LSA process. The service provides key process isolation ...	Started	Manual	Local System
COM+ Event System	Supports System Event Notification Service (SENS), which provides automatic distribution of events t...	Started	Automatic	Local Service
COM+ System Application	Manages the configuration and tracking of Component Object Model (COM)+-based components. I...		Manual	Local System
Computer Browser	Maintains an updated list of computers on the network and supplies this list to computers designate...	Started	Automatic	Local System
Cryptographic Services	Provides four management services: Catalog Database Service, which confirms the signatures of Win...	Started	Automatic	Network Service
DCOM Server Process Launcher	Provides launch functionality for DCOM services.	Started	Automatic	Local System
Desktop Window Manager Session Ma...	Provides Desktop Window Manager startup and maintenance services	Started	Automatic	Local System
DFS Replication	Replicates files among multiple PCs keeping them in sync. On Client, it is used to roam folders betwe...		Manual	Local System
DHCP Client	Registers and updates IP addresses and DNS records for this computer. If this service is stopped, this c...	Started	Automatic	Local Service
Diagnostic Policy Service	The Diagnostic Policy Service enables problem detection, troubleshooting and resolution for Window...	Started	Automatic	Local Service
Diagnostic Service Host	The Diagnostic Service Host service enables problem detection, troubleshooting and resolution for W...		Manual	Local Service
Diagnostic System Host	The Diagnostic System Host service enables problem detection, troubleshooting and resolution for W...	Started	Manual	Local System
Diskeeper	Controls the Windows Diskeeper Service	Started	Automatic	Local System
Distributed Link Tracking Client	Maintains links between NTFS files within a computer or across computers in a network.	Started	Automatic	Local System
Distributed Transaction Coordinator	Coordinates transactions that span multiple resource managers, such as databases, message queues, ...		Manual	Network Service
DNS Client	The DNS Client service (dnscache) caches Domain Name System (DNS) names and registers the full c...	Started	Automatic	Network Service
Extensible Authentication Protocol	The Extensible Authentication Protocol (EAP) service provides network authentication in such scenari...	Started	Manual	Local System
Fax	Enables you to send and receive faxes, utilizing fax resources available on this computer or on the net...		Manual	Network Service

Figure 39. Manage window services panel shows name, description, status, startup type and log on id of window services.


New Web Service

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Web Service Name: X3DWebServices

Project: SAVAGEWebServices

Location: Source Packages

Package: X3DWSMethod

(●) Create Web Service from Scratch

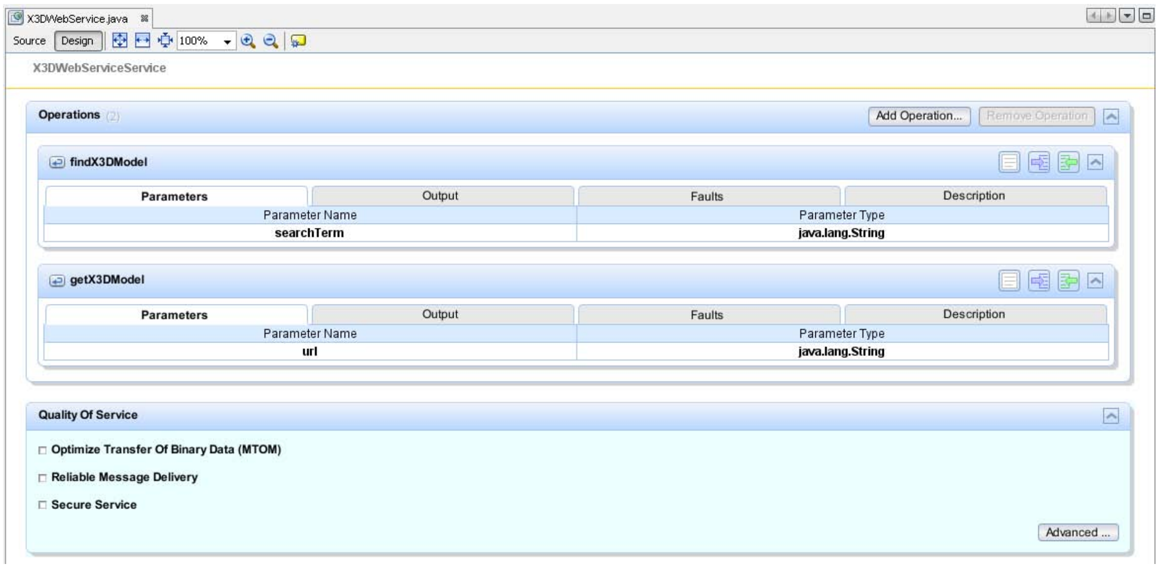
( ) Create Web Service from Existing Session Bean

Enterprise Bean:

Figure 40. Create web service using NetBeans web service wizard.

The NetBeans project "*SavageWebServices*" is created to contain the implementation of SAVAGE web services. It is a type of web application in NetBeans.

After this, right click the *SavageWebServices* web application and select "new web services". A "create web services" wizard will pop up. Fill in the web service name and the Java package it belongs to. Click "finish" and the web service is automatically generated.



X3DWebService.java

Source Design 100%

X3DWebServiceService

Operations (2) Add Operation... Remove Operation

findX3DModel

Parameters	Output	Faults	Description

Parameter Name	Parameter Type
searchTerm	java.lang.String

getX3DModel

Parameters	Output	Faults	Description

Parameter Name	Parameter Type
url	java.lang.String

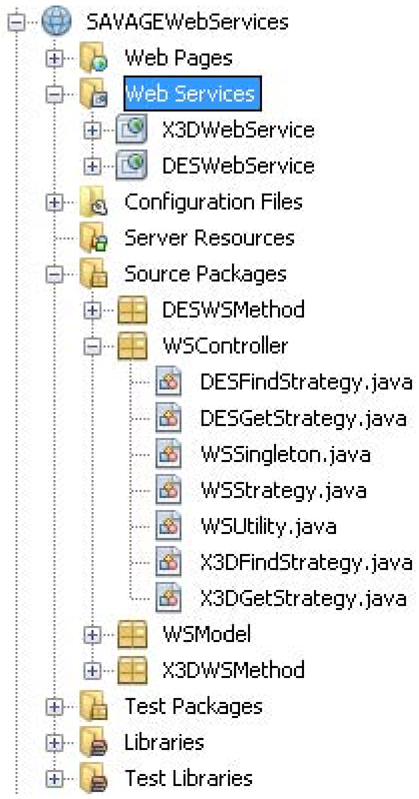
Quality Of Service

- ☐ Optimize Transfer Of Binary Data (MTOM)
- ☐ Reliable Message Delivery
- ☐ Secure Service

Advanced ...

Figure 41. NetBeans web service design view is used to add web service operations.

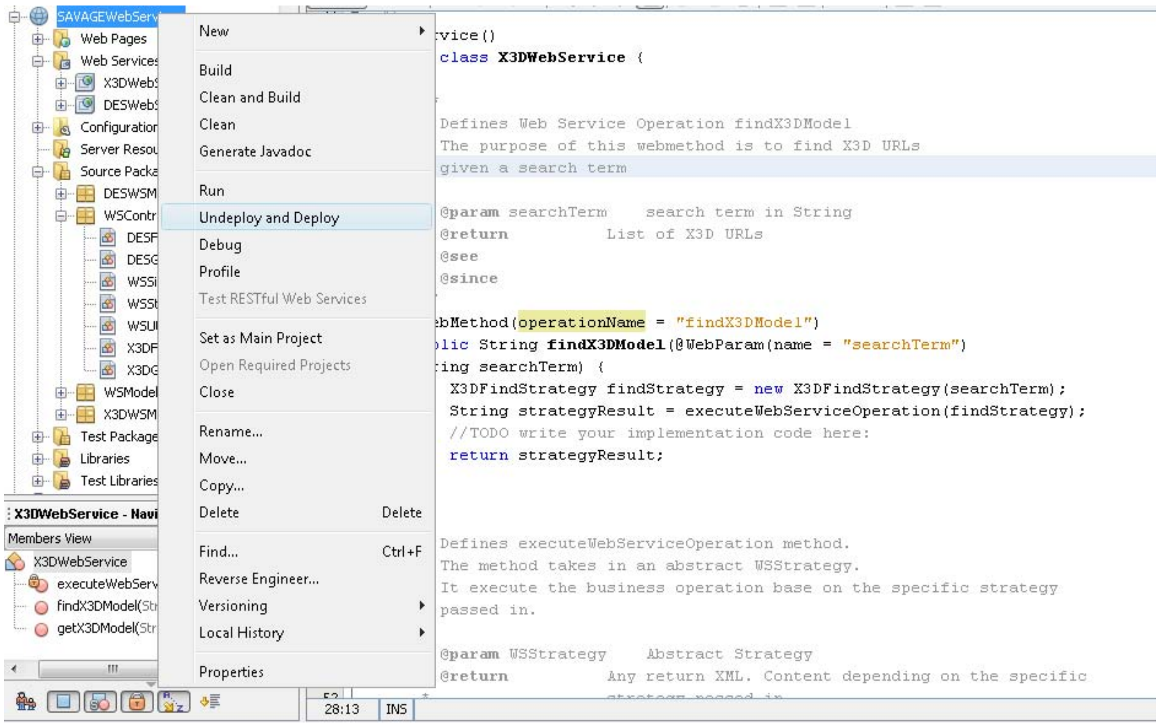
Use the visual designer to add web service operations (Figure 41). The source code for the web method will be automatically generated.



```
SAVAGEWebServices
    Web Pages
    Web Services
        X3DWebService
        DESWebService
    Configuration Files
    Server Resources
    Source Packages
        DESWSMethod
        WSController
            DESFindStrategy.java
            DESGetStrategy.java
            WSSingleton.java
            WSStrategy.java
            WSUtility.java
            X3DFindStrategy.java
            X3DGetStrategy.java
        WSModel
        X3DWSMethod
    Test Packages
    Libraries
    Test Libraries
```

Figure 42. NetBeans project view gives a good overview of projects, software components, library, configuration files and source code.

Taking reference to the SAVAGE web services design specified in Chapter III, implement the source code for the web methods in *X3DWebService* and *DESWebService*. The source code for the various classes in *DESWSMethod*, *X3DWSMethod*, *WSController* and *WSModel* Java packages is also implemented according to what is specified in the design specification. The layout of the created packages, classes and web services is shown in Figure 42.

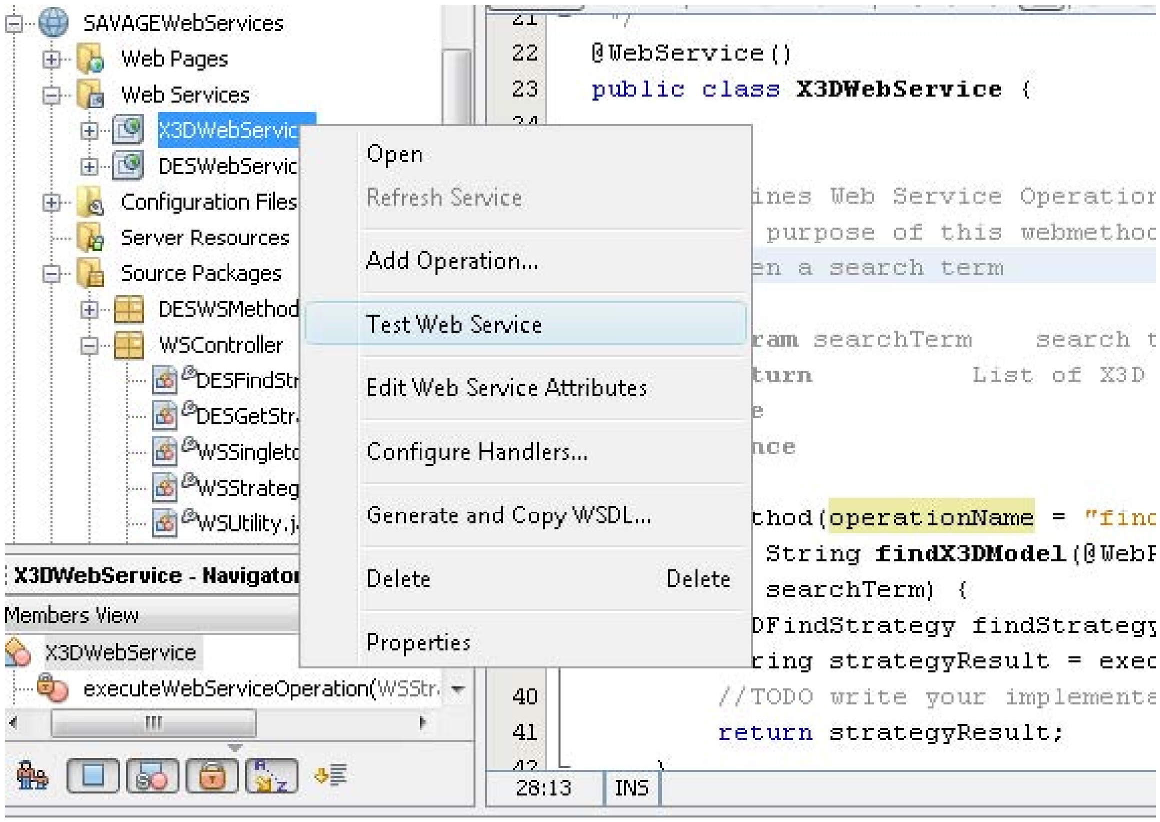


```
:vice()
class X3DWebService {

Defines Web Service Operation findX3DModel
The purpose of this webmethod is to find X3D URLs
given a search term

@param searchTerm    search term in String
@return          List of X3D URLs
@see
@since

ebMethod(operationName = "findX3DModel")
plic String findX3DModel(@WebParam(name = "searchTerm")
ting searchTerm) {
  X3DFindStrategy findStrategy = new X3DFindStrategy(searchTerm);
  String strategyResult = executeWebServiceOperation(findStrategy);
  //TODO write your implementation code here:
  return strategyResult;

Defines executeWebServiceOperation method.
The method takes in an abstract WSStrategy.
It execute the business operation base on the specific strategy
passed in.

@param WSStrategy    Abstract Strategy
@return          Any return XML. Content depending on the specific
```

Figure 43. Undeploy and deploy web services using NetBeans.

To build and deploy Savage web services, right click *SavageWebService* web application and select "build from menu". Upon successful build, select Undeploy and Deploy from the menu (Figure 43). Ensure that the Tomcat server is running before invoking the deploy option.

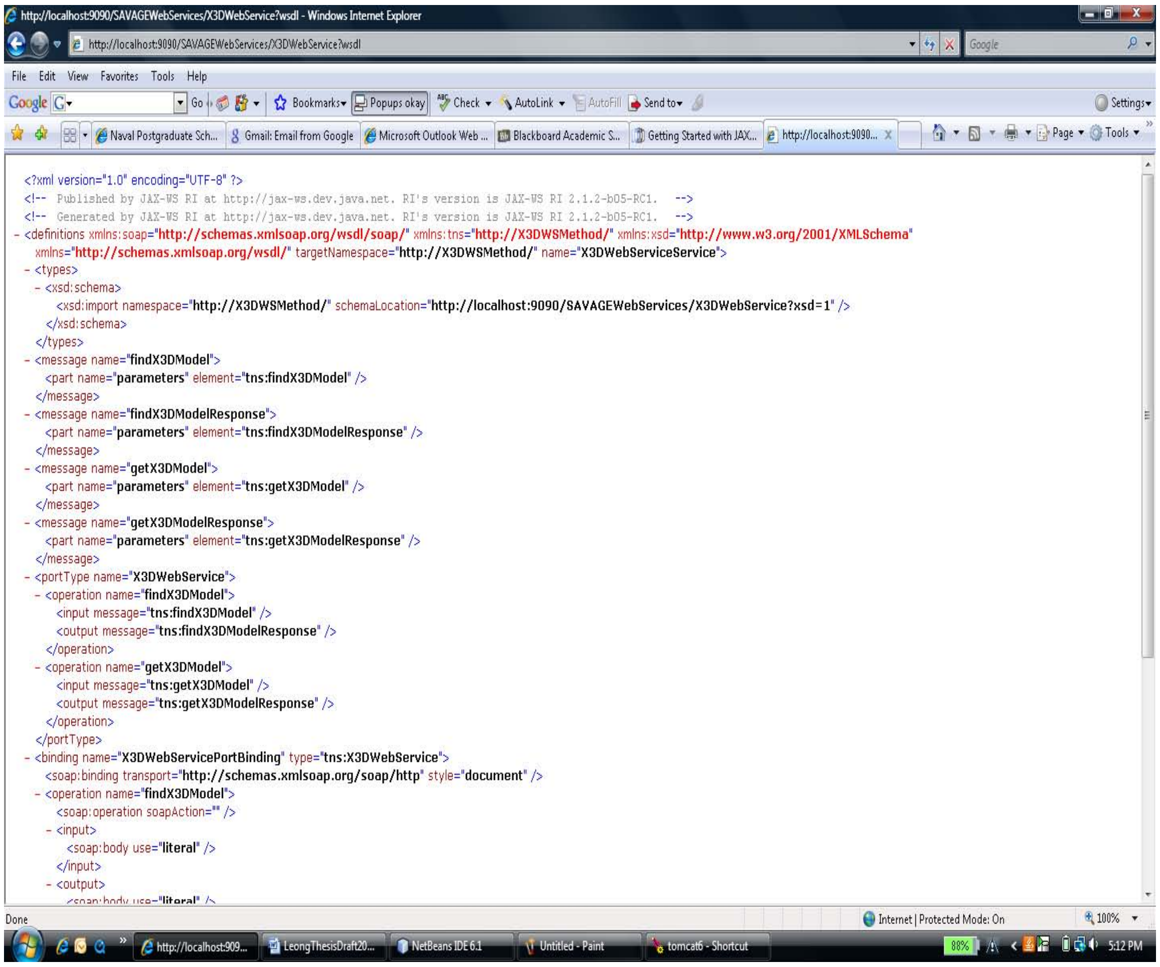
Should there be complications in the build and deployment process, select "clean from menu". Confirm that the application is undeployed on the Tomcat application server. Restart the Tomcat server. Perform the build and deploy process again.



```
@WebService()
public class X3DWebService {

ines Web Service Operatior
 purpose of this webmethoc
en a search term

ram searchTerm    search t
turn              List of X3D
e
nce

thod(operationName = "finc
 String findX3DModel(@WebF
 searchTerm) {
DFindStrategy findStrategy
ring strategyResult = exec
    //TODO write your implementa
    return strategyResult;
```

Figure 44. Test web services in NetBeans.

Right click the deployed web service and select Test Web Service from menu. In the example shown in figure 44, the Internet Explorer (IE) browser is automatically launched as client. The IE browser then invokes the *X3DWebService*.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.2-b05-RC1.  -->
<!--  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.2-b05-RC1.  -->
- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://X3DWSMethod/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://X3DWSMethod/" name="X3DWebServiceService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://X3DWSMethod/" schemaLocation="http://localhost:9090/SAVAGEWebServices/X3DWebService?xsd=1" />
      </xsd:schema>
    </types>
  - <message name="findX3DModel">
      <part name="parameters" element="tns:findX3DModel" />
    </message>
  - <message name="findX3DModelResponse">
      <part name="parameters" element="tns:findX3DModelResponse" />
    </message>
  - <message name="getX3DModel">
      <part name="parameters" element="tns:getX3DModel" />
    </message>
  - <message name="getX3DModelResponse">
      <part name="parameters" element="tns:getX3DModelResponse" />
    </message>
  - <portType name="X3DWebService">
    - <operation name="findX3DModel">
        <input message="tns:findX3DModel" />
        <output message="tns:findX3DModelResponse" />
      </operation>
    - <operation name="getX3DModel">
        <input message="tns:getX3DModel" />
        <output message="tns:getX3DModelResponse" />
      </operation>
    </portType>
  - <binding name="X3DWebServicePortBinding" type="tns:X3DWebService">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    - <operation name="findX3DModel">
        <soap:operation soapAction="" />
      - <input>
          <soap:body use="literal" />
        </input>
      - <output>
          <soap:body use="literal" />
```

Figure 45. WSDL file that describes X3DWebService.

The WSDL generated by JAX-WS is as shown in Figure 45. JAX-WS is part of the J2EE platform from Sun Microsystems.

C. DEPLOYMENT

There are two main application servers used for the deployment (see Figure 46). First is the APACHE web server. External clients make https invocations to this server. The confidentiality of messages that are sent between clients and the APACHE web server is ensured at the transport layer via SSL (port 443) encryption. The APACHE web server also plays the role as proxy or façade to other SAVAGE data and application servers. The motivation is to protect SAVAGE servers from external clients. SAVAGE web services are deployed on a Tomcat web application server. The APACHE web server redirects the web service invocation request to the Tomcat web application server. The messages that are sent between SAVAGE servers are not encrypted for performance reasons.

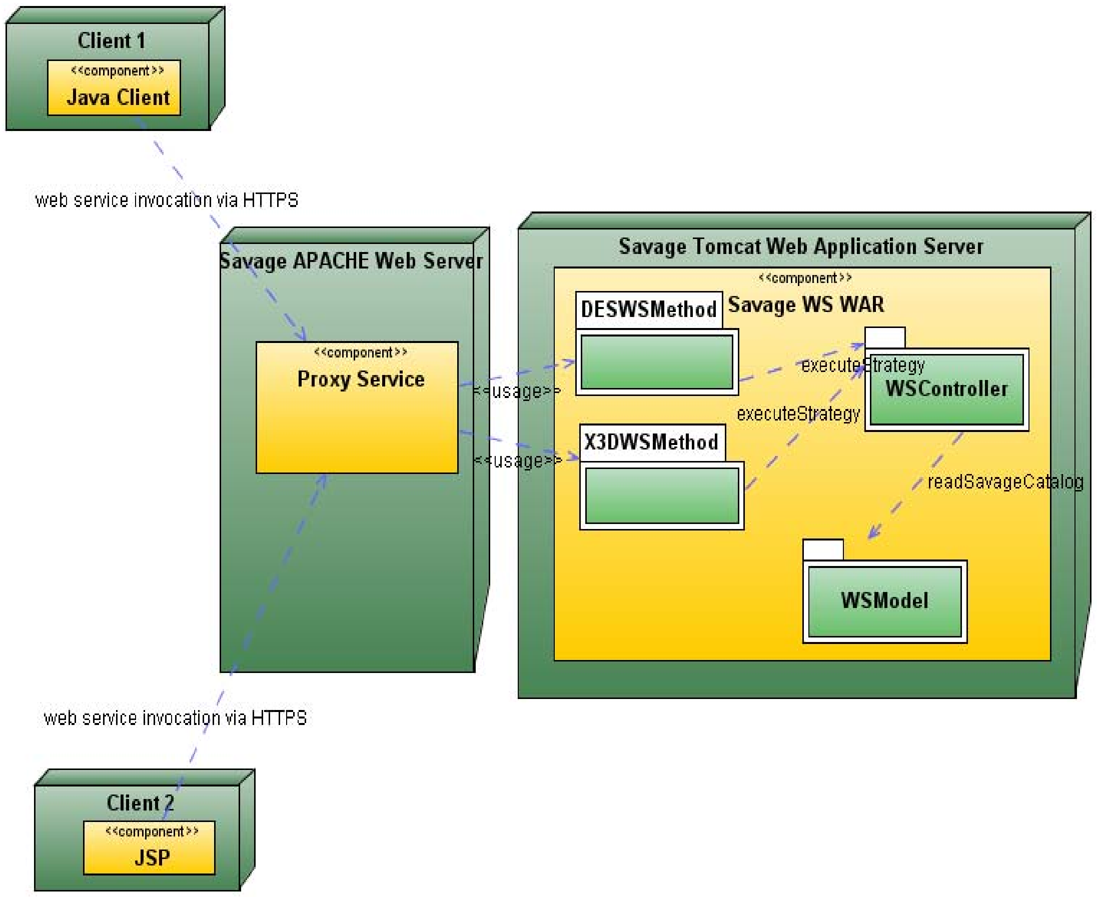


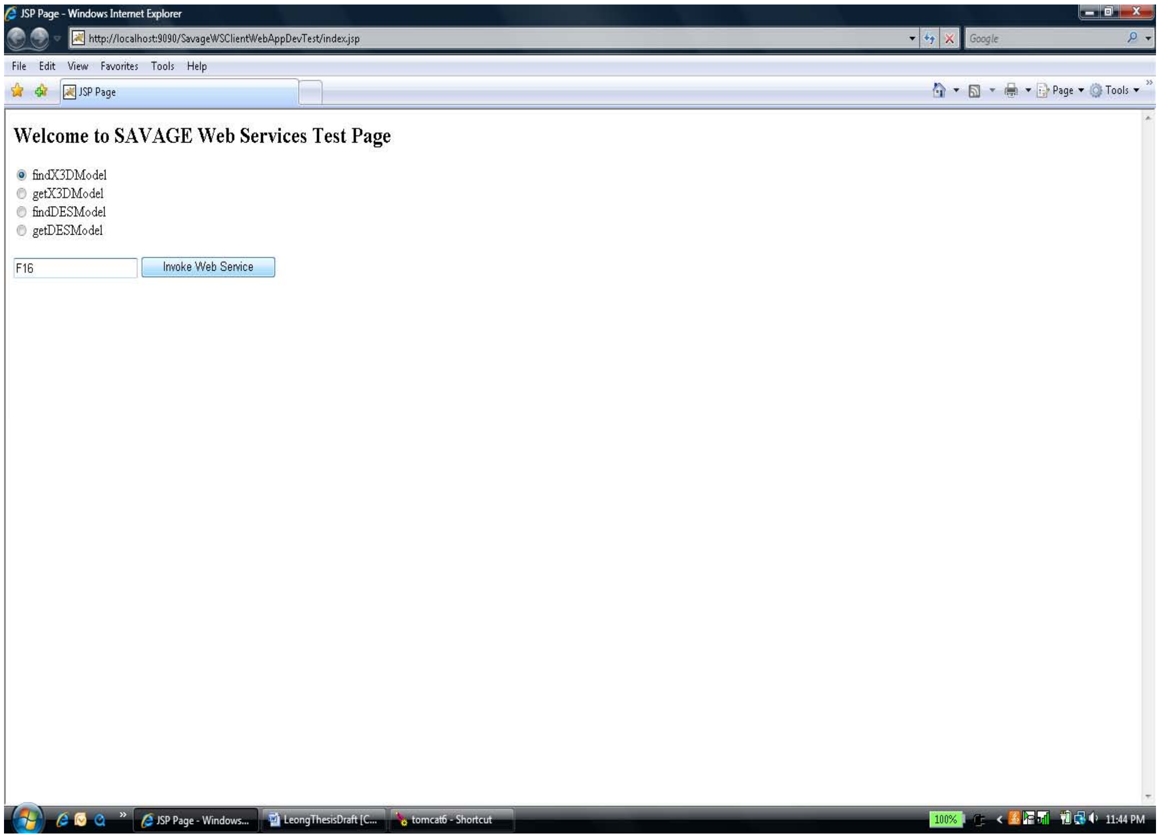
Figure 46. SAVAGE web services deployment diagram in UML. Web services are deployed in SAVAGE Tomcat Web Application Server.

SAVAGE web services are tested and compiled using the NetBeans IDE on a development machine. The Java war file is sent from the development machine via Secure File Transfer Protocol (SFTP) to the Tomcat web application server /USR/JAVA/TOMCAT/WEBAPPS/ directory. Tomcat web application server log files are found in the /usr/java/apache-tomcat-5.5.17/logs directory.

D. TEST CLIENT SET UP

Create a separate project of type “Java application for Java client”. The Java client is required to test the web methods. The Java client can be Java classes leveraging JAX-WS to invoke SAVAGE web services. Alternatively, web services can be invoked via the

Java Servlet Page (JSP). A demonstration JSP client to invoke SAVAGE web services was developed. The user selects the web method via the associated radio button and keys in input parameters in the text box. The user then clicks the "Invoke Web Service" button (see Figure 47). The selected web method is invoked. The client receives and writes the XML result to a file. A hyperlink named "Savage Web Service Invocation Result" to the file is automatically created on JSP (Figure 48). The user clicks on the hyperlink and the XML result is displayed in a separate browser window. For an X3D model, the .x3d file extension is automatically detected by the browser. Thus, it can be automatically displayed with an X3D compatible viewer such as Xj3D Viewer. The list of X3D resources is found in http://www.web3d.org/x3d/content/examples/X3dResources.html.

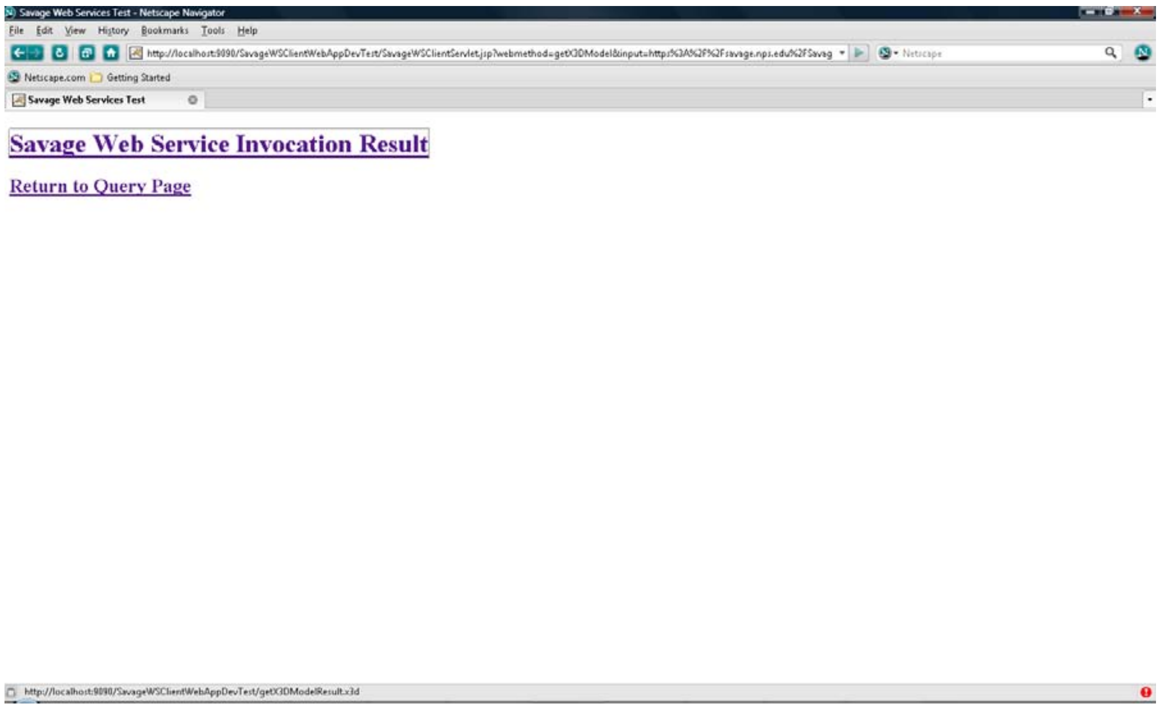

JSP Page - Windows Internet Explorer
http://localhost:9090/SavageWSClientWebAppDevTest/index.jsp
File Edit View Favorites Tools Help
JSP Page
Page Tools

Welcome to SAVAGE Web Services Test Page

- [✓ findX3DModel]
- getX3DModel
- findDESModel
- getDESModel

F16 Invoke Web Service

Figure 47. SAVAGE web services JSP test page. The user selects the web method, keys in parameters and clicks "Invoke Web Service".


Savage Web Services Test - Netscape Navigator
File Edit View History Bookmarks Tools Help
Netscape
Netscape.com
Getting Started
Savage Web Services Test

http://localhost:9090/SavageWSClientWebAppDevTest/SavageWSClientServlet.jsp?webmethod=getX3DModel&input=https%3A%2F%2Fsavage.nps.edu%2FSavag

Savage Web Service Invocation Result

Return to Query Page

http://localhost:9090/SavageWSClientWebAppDevTest/getX3DModelResult.x3d

Figure 48. JSP test page that contains hyperlink to SAVAGE web service invocation result.

NetBeans makes establishing the web services client easy. The developer right clicks on "project" and selects "new→web service client" (Figure 49). A wizard pops up and prompts for the location of the WSDL (Figure 50). The WSDL locations for X3D and DES web services in the development environment are found in http://localhost:9090/SAVAGEWebServices/X3DWebService?wsdl and http://localhost:9090/SAVAGEWebServices/DESWebService?wsdl, respectively. The WSDL locations for X3D and DES web services in the production environment are found in https://savage.nps.edu/SAVAGEWebServices/X3DWebService?wsdl and https://savage.nps.edu/SAVAGEWebServices/DESWebService?wsdl, respectively. Upon indicating the location of WSDL, NetBeans reads in the WSDL file to gather the operations, endpoint and binding for SAVAGE web services. This information is used to generate necessary Java classes which incorporate JAX-WS and other necessary Java APIs to establish the web services connectivity. The code required to call these Java classes can be easily generated by NetBeans and incorporated (via drag and drop) into a Java main class or JSP. The Java main class or JSP will trigger the invocation of web services when the program runs.

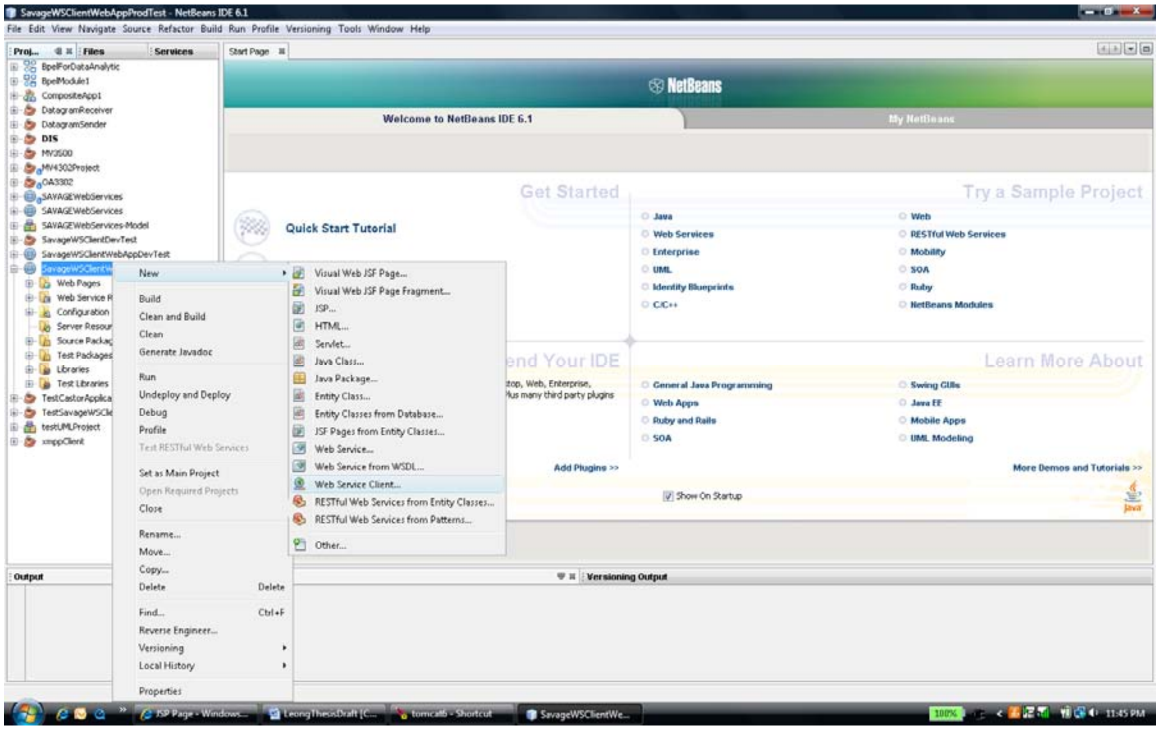
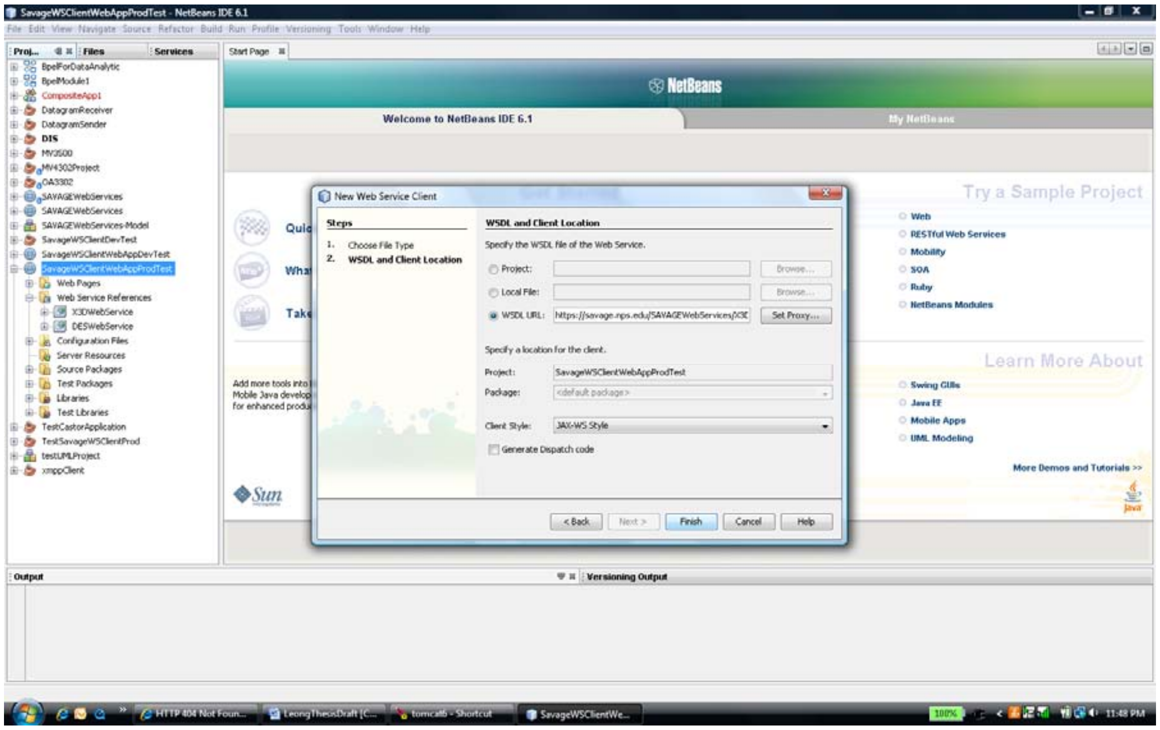


Figure 49. Creating web services client in NetBeans.



Specify the WSDL file of the Web Service.

Project:

Local File:

WSDL URL: https://savage.nps.edu/SAVAGEWebServices/X3C

Specify a location for the client.

Project: SavageWSClientWebAppProdTest

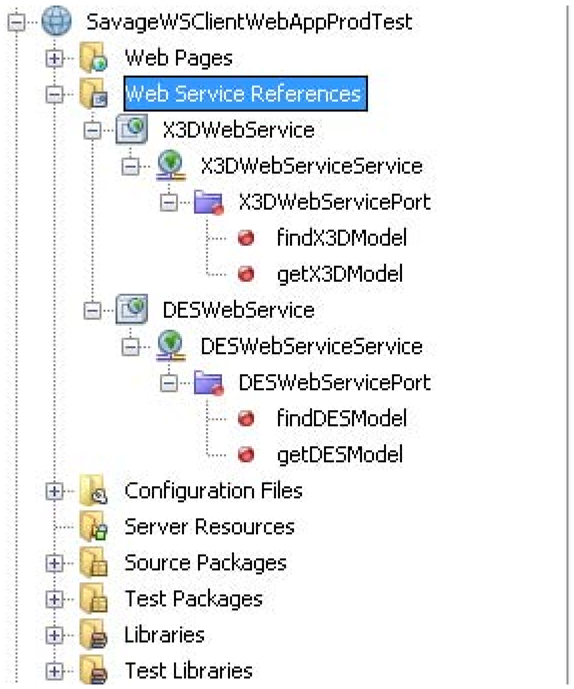
Package: <default package>

Client Style: JAX-WS Style

Generate Dispatch code

Figure 50. Web service client wizard is used to create the web service client that facilitates connectivity to SAVAGE web services in NetBeans.

Once the web service client connectivity is established, NetBeans will respond by generating and displaying "web service references" on the IDE (Figure 51). These are Java objects that capture information about the web services.



```
SavageWSClientWebAppProdTest
  Web Pages
  Web Service References
    X3DWebService
      X3DWebServiceService
        X3DWebServicePort
          findX3DModel
          getX3DModel
    DESWebService
      DESWebServiceService
        DESWebServicePort
          findDESModel
          getDESModel
  Configuration Files
  Server Resources
  Source Packages
  Test Packages
  Libraries
  Test Libraries
```

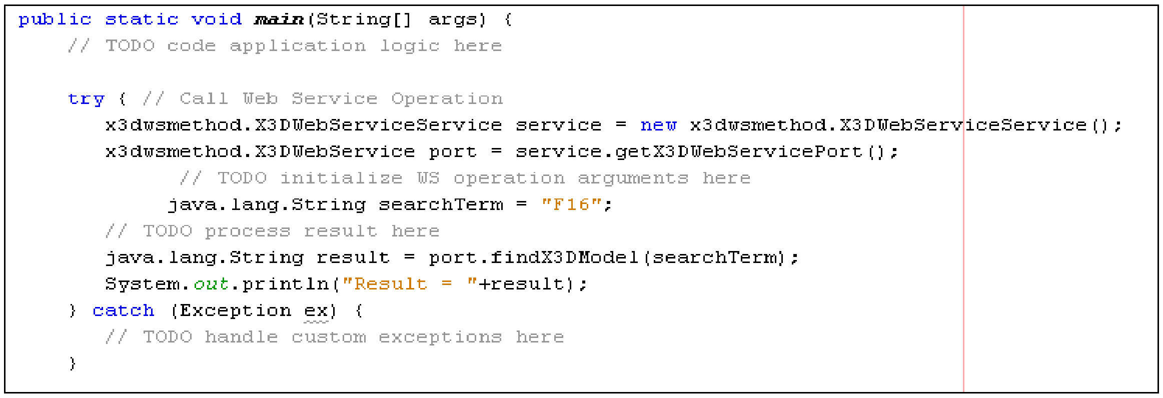
Figure 51. Web services client established in NetBeans. Web service references contain web service client objects which can be connected to a web service.

E. TEST RESULTS

Test plans were established for web methods *findX3DModel*, *getX3DModel*, *findDESModel* and *getDESModel*. Tests were executed according to the test plan within the implementation setup. All test results verified correct. The details of the test plan and result for each web method are elaborated below.

a. *findX3DModel Webmethod*

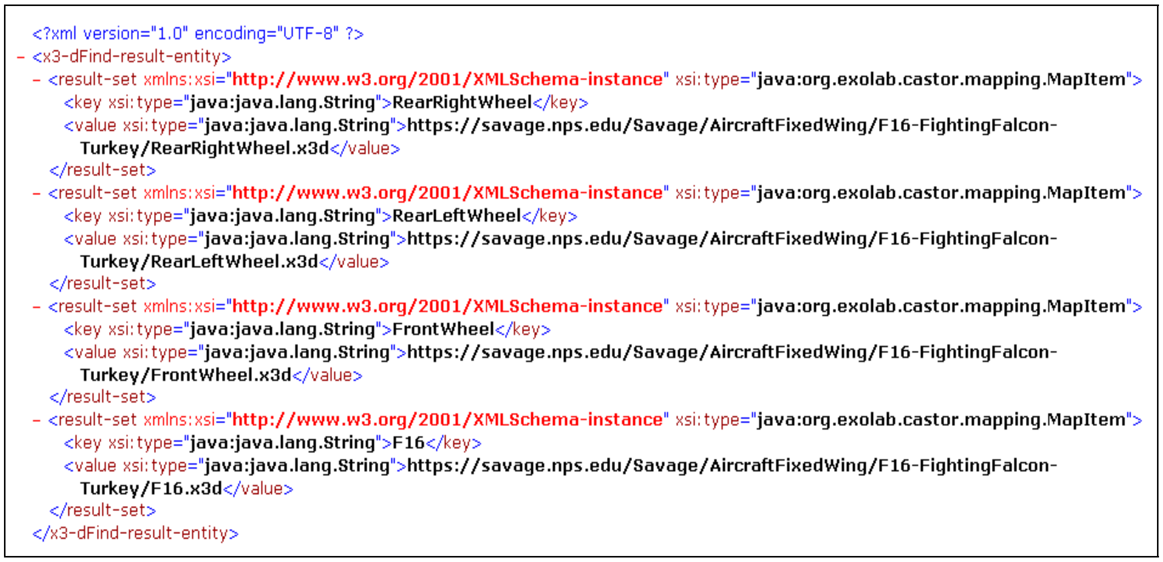
Test plan – Java client invokes *X3DWebService findX3DModel* web method. The search term (input parameter) is "F16". *X3DWebService* receives the request, processes the search term (performing string match test with the content of name, URL and description attributes) and returns list of X3D names and URLs with attributes that match the search term.



```
public static void main(String[] args) {
    // TODO code application logic here

    try { // Call Web Service Operation
       x3dwsmethod.X3DWebServiceService service = new x3dwsmethod.X3DWebServiceService();
       x3dwsmethod.X3DWebService port = service.getX3DWebServicePort();
             // TODO initialize WS operation arguments here
            java.lang.String searchTerm = "F16";
       // TODO process result here
       java.lang.String result = port.findX3DModel(searchTerm);
       System.out.println("Result = "+result);
    } catch (Exception ex) {
       // TODO handle custom exceptions here
    }
```

Figure 52. Source code that invokes *findX3DModel* web method in Java client main class.

The code snippet that implements the Java client for this test plan is shown in Figure 52. The search term “F16” is captured in variable *searchTerm* and the result string (list of matching X3D names and URLs) is captured in variable *result*. The result is printed out on the console.



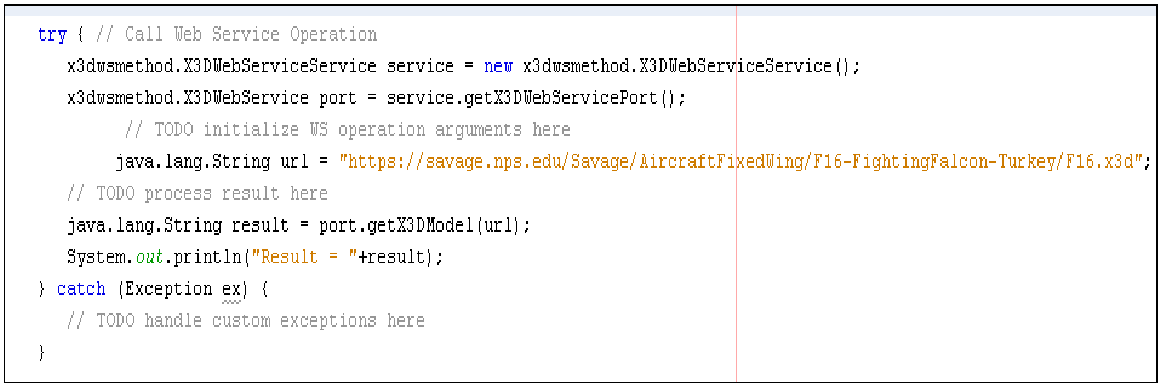
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <x3-dFind-result-entity>
  - <result-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="java:org.exolab.castor.mapping.MapItem">
      <key xsi:type="java:java.lang.String">RearRightWheel</key>
      <value xsi:type="java:java.lang.String">https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-
        Turkey/RearRightWheel.x3d</value>
    </result-set>
  - <result-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="java:org.exolab.castor.mapping.MapItem">
      <key xsi:type="java:java.lang.String">RearLeftWheel</key>
      <value xsi:type="java:java.lang.String">https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-
        Turkey/RearLeftWheel.x3d</value>
    </result-set>
  - <result-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="java:org.exolab.castor.mapping.MapItem">
      <key xsi:type="java:java.lang.String">FrontWheel</key>
      <value xsi:type="java:java.lang.String">https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-
        Turkey/FrontWheel.x3d</value>
    </result-set>
  - <result-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="java:org.exolab.castor.mapping.MapItem">
      <key xsi:type="java:java.lang.String">F16</key>
      <value xsi:type="java:java.lang.String">https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-
        Turkey/F16.x3d</value>
    </result-set>
  </x3-dFind-result-entity>
```

Figure 53. Return result for *FindX3DModel* in XML.

The result XML is shown in Figure 53. The return result includes X3D names and URLs of F16 and its related aircraft components (RearLeftWheel, RearRightWheel and FrontWheel). All the models are retrieved from the file location https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-Turkey/. Hence, the result is precisely correct.

b. *getX3DModel Webmethod*

Test plan – Java client invokes *getX3DModel* web method from *X3DWebService*. X3D URL “https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-Turkey/F16.x3d” is the input parameter. *X3DWebService* will receive the request, go to the specific file location in Savage repository and retrieve the X3D model. The retrieved X3D model is returned to the Java client. The Java client displays the X3D model on console.



```
try { // Call Web Service Operation
   x3dwsmethod.X3DWebServiceService service = new x3dwsmethod.X3DWebServiceService();
   x3dwsmethod.X3DWebService port = service.getX3DWebServicePort();
         // TODO initialize WS operation arguments here
        java.lang.String url = "https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-Turkey/F16.x3d";
   // TODO process result here
   java.lang.String result = port.getX3DModel(url);
   System.out.println("Result = "+result);
} catch (Exception ex) {
   // TODO handle custom exceptions here
}
```

Figure 54. Source code that invokes *getX3DModel* web method in java client main class.

The code snippet in Figure 54 implements the Java client for this test plan. The X3D URL “https://savage.nps.edu/Savage/AircraftFixedWing/F16-FightingFalcon-Turkey/F16.x3d” is captured in variable *URL* and the result string (X3D model) is captured in variable *result*. The result displayed by an X3D-compatible viewer is shown in Figure 55.

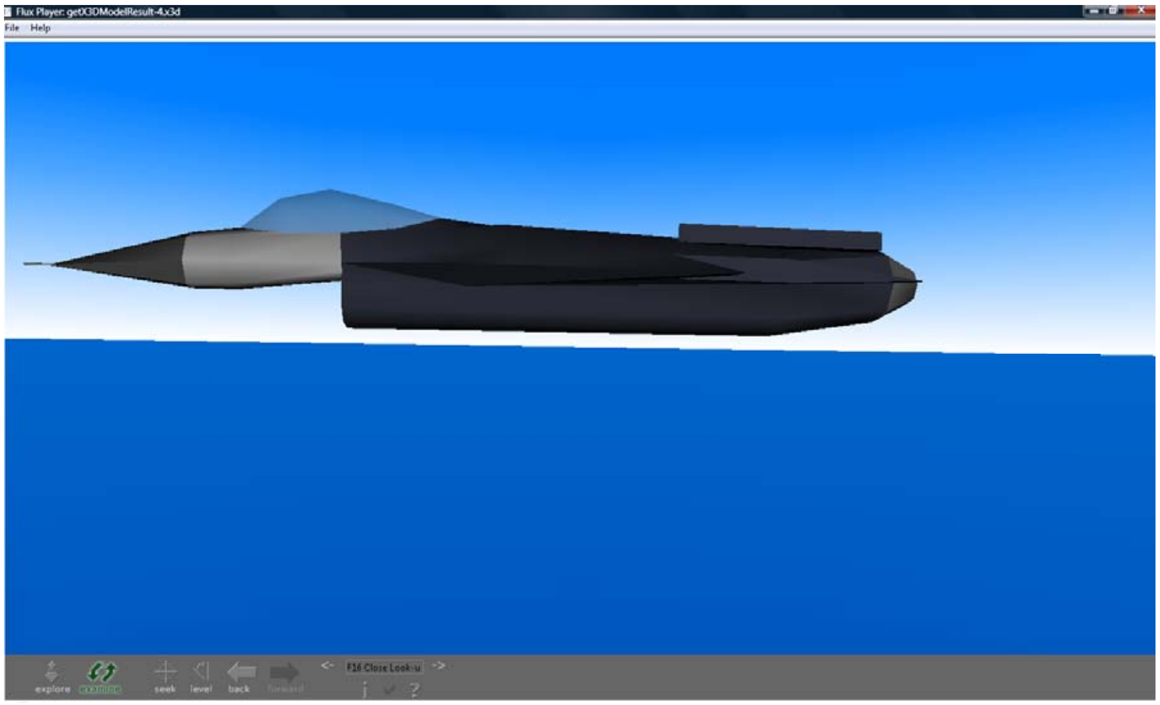
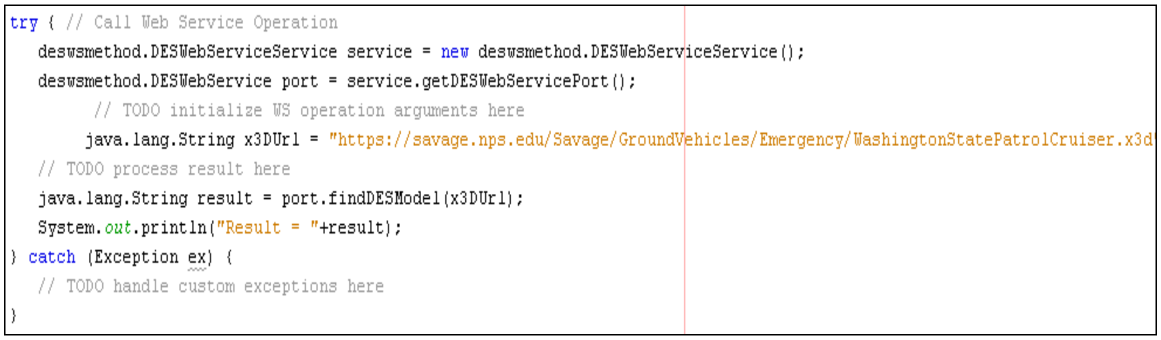


Figure 55. Returned X3D model for *getX3DModel* web method. The X3D model is in XML and is displayed by an X3D-compatible viewer.

One of the metadata elements that the X3D model describes is the identifier containing the URL of the X3D model. The URL is the same as the input parameter (X3D URL) passed in by the Java client. Hence, the result X3D model displayed on the console is correct.

c. *findDESModel Webmethod*

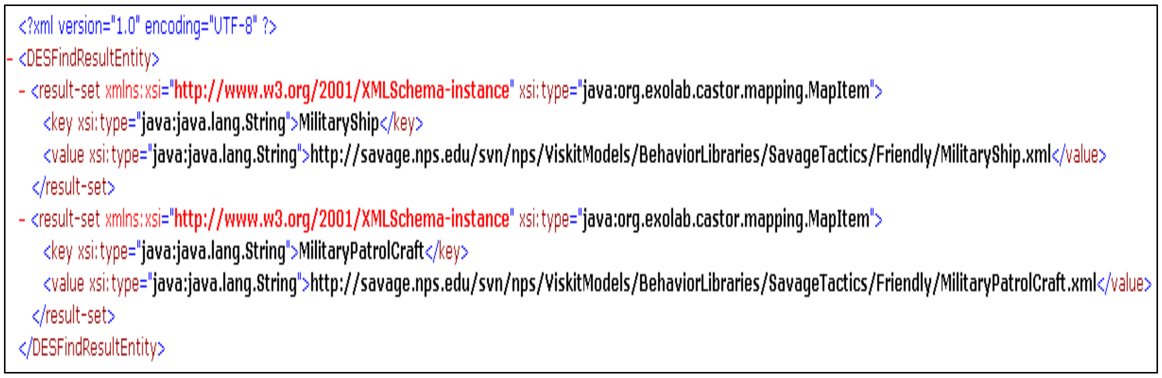
Test plan - Java client invokes *findDESModel* web method from *DESWebService*. X3D URL "https://savage.nps.edu/SAVAGE/GroundVehicles/Emergency/WashingtonStatePatrolCruiser.x3d" is the input parameter. *X3DDESWebService* receives the web service request, searches the SAVAGE catalog (specifically *SimulationAgent* under *SMAL*) for the matching DES behavior, extracts a list of DES agent and URL, and parses the information to XML before sending it back to the Java client.



```
try { // Call Web Service Operation
    deswsmethod.DESWebServiceService service = new deswsmethod.DESWebServiceService();
    deswsmethod.DESWebService port = service.getDESWebServicePort();
        // TODO initialize WS operation arguments here
        java.lang.String x3DUrl = "https://savage.nps.edu/Savage/GroundVehicles/Emergency/WashingtonStatePatrolCruiser.x3d
    // TODO process result here
    java.lang.String result = port.findDESModel(x3DUrl);
    System.out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
```

Figure 56. Source code that invokes *findDESModel* web method in java client main class.

The code snippet that implements the Java client is shown in Figure 56. Input parameter (X3D URL) "https://savage.nps.edu/Savage/GroundVehicles/Emergency/WashingtonStatePatrolCruiser.x3d" is captured in variable *x3DURL*. The output parameter (list of DES URLs and agent in XML) is captured in variable *result*. The result string is displayed on the console (Figure 57).



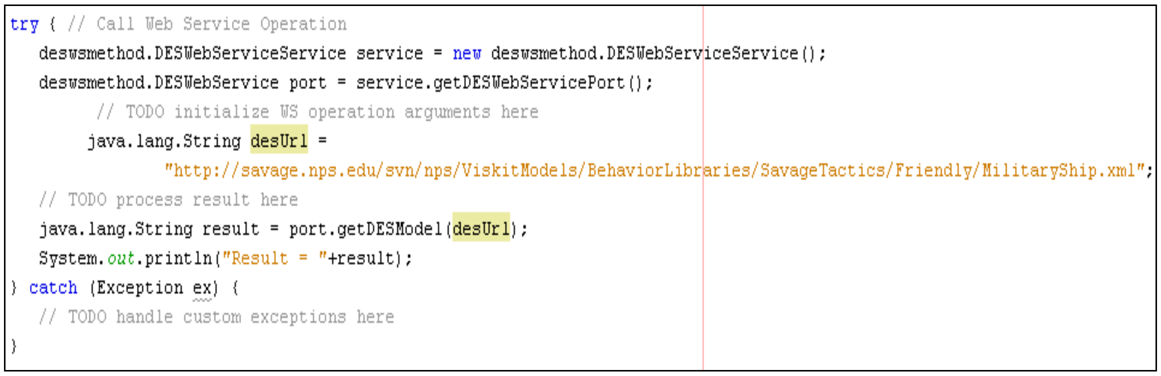
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <DESFindResultEntity>
  - <result-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="java:org.exolab.castor.mapping.MapItem">
      <key xsi:type="java:java.lang.String">MilitaryShip</key>
      <value xsi:type="java:java.lang.String">http://savage.nps.edu/svn/nps/ViskitModels/BehaviorLibraries/SavageTactics/Friendly/MilitaryShip.xml</value>
    </result-set>
  - <result-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="java:org.exolab.castor.mapping.MapItem">
      <key xsi:type="java:java.lang.String">MilitaryPatrolCraft</key>
      <value xsi:type="java:java.lang.String">http://savage.nps.edu/svn/nps/ViskitModels/BehaviorLibraries/SavageTactics/Friendly/MilitaryPatrolCraft.xml</value>
    </result-set>
  </DESFindResultEntity>
```

Figure 57. Return result for *findDESModel* web method in XML.

The result corresponds to the set of DES behaviors described in the SAVAGE catalog (i.e., given a matching X3D URL in the *page* element, the list of DES behaviors is found in *SMAL->BehaviorParameterSet->SimulationAgent* element). Hence, the result is correct.

d. getDESModel Webmethod

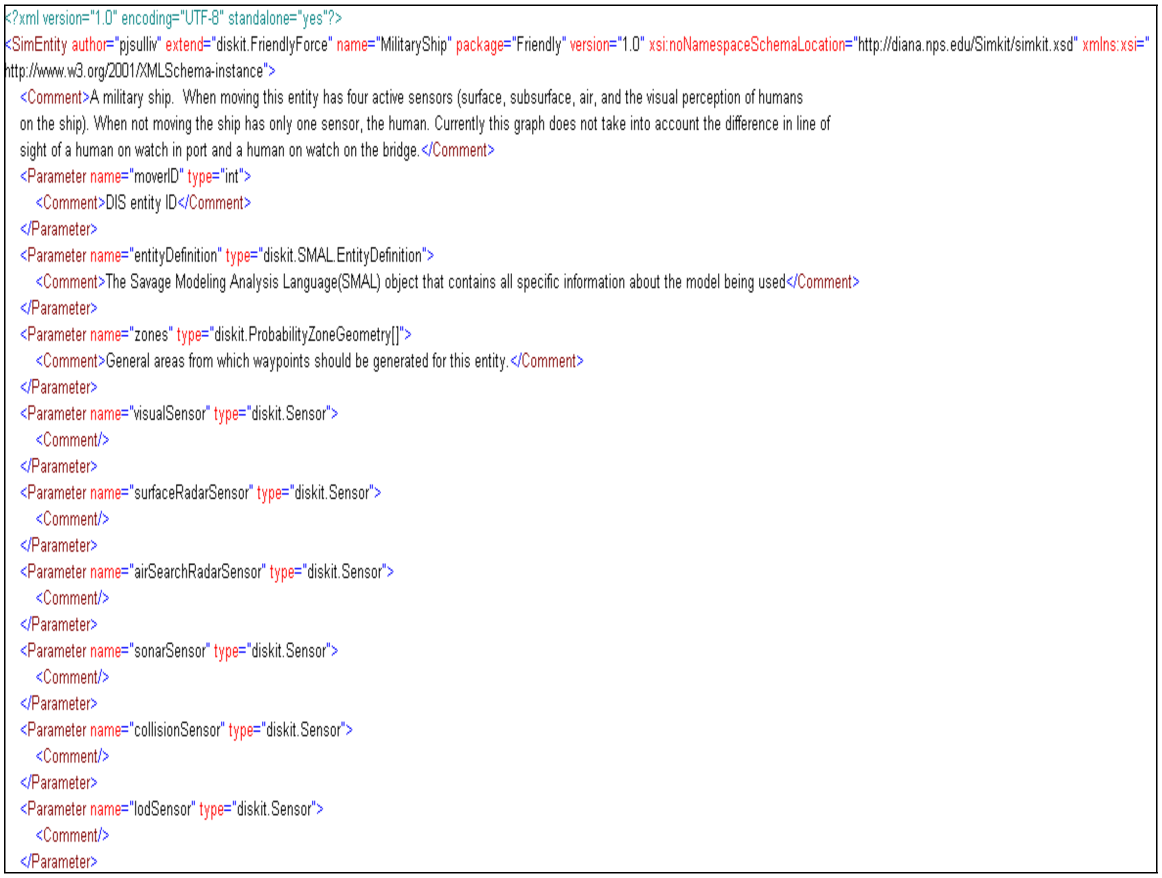
Test plan - Java client sends a request to *getDESModel* web method from *DESWebService*. DES URL "http://savage.nps.edu/svn/nps/ViskitModels/BehaviorLibraries/SavageTactics/Friendly/MilitaryShip.xml" is the input parameter. *DESWebService* receives the request and retrieves the DES model specified by the URL. The DES model is sent back to the Java client.



```
try { // Call Web Service Operation
   deswsmethod.DESWebServiceService service = new deswsmethod.DESWebServiceService();
   deswsmethod.DESWebService port = service.getDESWebServicePort();
         // TODO initialize WS operation arguments here
        java.lang.String desUrl =
                "http://savage.nps.edu/svn/nps/ViskitModels/BehaviorLibraries/SavageTactics/Friendly/MilitaryShip.xml";
   // TODO process result here
   java.lang.String result = port.getDESModel(desUrl);
   System.out.println("Result = "+result);
} catch (Exception ex) {
   // TODO handle custom exceptions here
}
```

Figure 58. Source code that invokes *getDESModel* web method in java client main class.

Figure 58 show a code snippet of the Java client implementation. *DesURL* stores the input parameter and variable *result* stores the DES model. The result string is displayed on the console (Figure 59).



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SimEntity author="pjsulliv" extend="diskit.FriendlyForce" name="MilitaryShip" package="Friendly" version="1.0" xsi:noNamespaceSchemaLocation="http://diana.nps.edu/Simkit/simkit.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
  <Comment>A military ship.  When moving this entity has four active sensors (surface, subsurface, air, and the visual perception of humans
  on the ship). When not moving the ship has only one sensor, the human. Currently this graph does not take into account the difference in line of
  sight of a human on watch in port and a human on watch on the bridge.</Comment>
  <Parameter name="moverID" type="int">
    <Comment>DIS entity ID</Comment>
  </Parameter>
  <Parameter name="entityDefinition" type="diskit.SMAL.EntityDefinition">
    <Comment>The Savage Modeling Analysis Language(SMAL) object that contains all specific information about the model being used</Comment>
  </Parameter>
  <Parameter name="zones" type="diskit.ProbabilityZoneGeometry[]">
    <Comment>General areas from which waypoints should be generated for this entity.</Comment>
  </Parameter>
  <Parameter name="visualSensor" type="diskit.Sensor">
    <Comment/>
  </Parameter>
  <Parameter name="surfaceRadarSensor" type="diskit.Sensor">
    <Comment/>
  </Parameter>
  <Parameter name="airSearchRadarSensor" type="diskit.Sensor">
    <Comment/>
  </Parameter>
  <Parameter name="sonarSensor" type="diskit.Sensor">
    <Comment/>
  </Parameter>
  <Parameter name="collisionSensor" type="diskit.Sensor">
    <Comment/>
  </Parameter>
  <Parameter name="lodSensor" type="diskit.Sensor">
    <Comment/>
  </Parameter>
```

Figure 59. Return result for *getDESModel* web method in XML.

The result DES model shows package=”Friendly” and name=”MilitaryShip”. This corresponds to “Friendly/MilitaryShip.xml” from the input variable. Thus, the DES model retrieved is correct.

F. SUMMARY

This chapter elaborates on the implementation setup on the development environment. The chapter then elaborates on the deployment of SAVAGE web services. The JSP client is required to establish connection and invoke SAVAGE web services. The chapter rounds up by describing the test plan and results for each web method (*findX3DModel*, *getX3DModel*, *findDESModel* and *getDESModel*). Each web method corresponds to its use case described in chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. WSAIF ORCHESTRATION AND ADAPTATION

A. INTRODUCTION

This chapter first talks about a SAVAGE orchestration scenario. WSBPEL and OWL-S are used to model the orchestration scenario. A comparison between WSBPEL and OWL-S is made based on observation of the SAVAGE orchestration scenario. Comparison between WSMO and OWL-S is summarized from the literature. The chapter ends by explaining the approach to integrate software agents and data models to enable web services integration on the fly.

B. WS ORCHESTRATION SCENARIO FOR SAVAGE WEB SERVICES

Web method *findX3DModel* will invoke web method *getX3DModel* or *findDESModel* depending on an *if* condition passed in when the process is triggered. If the *findDESModel* web method is triggered, then web method *getDESModel* will be triggered sequentially to retrieve the associated DES behavior model.

C. SAVAGE WS ORCHESTRATION AND ADAPTATION USING WSBPEL

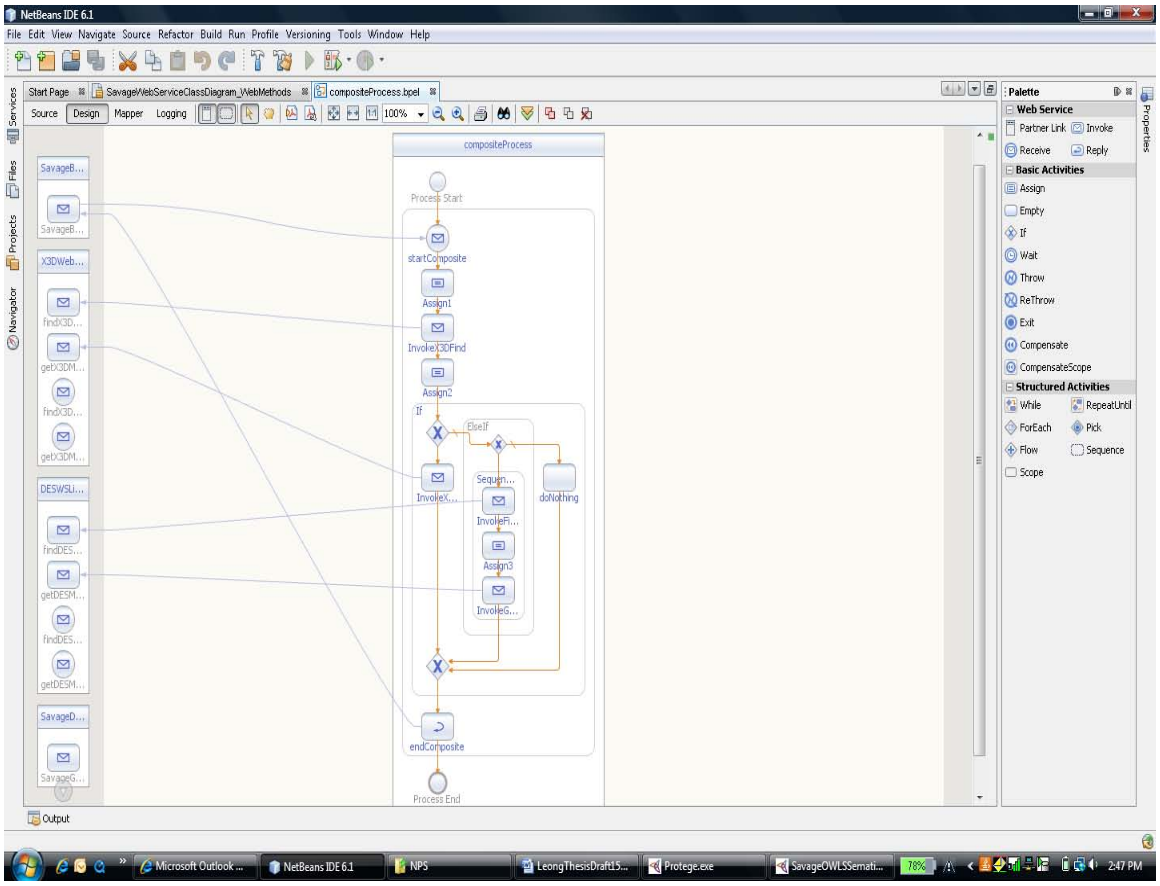


Figure 60. SAVAGE WSBPEL composite process in NetBeans design view. The composite process includes SAVAGE web services methods.

NetBeans provides a tool to create WSBPEL (see Figure 60). NetBeans classifies WSBPEL activities into three types. The web service activities type includes invocation, receive, reply and partner link to partner web services. The basic activities type comprises essential constructs that are required throughout the composite workflow. Examples are *if*, *wait* (wait for an indicated period of time), *empty* (do nothing), and *assign* (mapping between variables). The structure activities type is able to group or restrict activities within a composite workflow. Examples of the structure activities type are *While*, *ForEach*, *RepeatUntil*, and *Sequence*.

It is simple to create a WSBPEL model using the project creation wizard. Just select new project, SOA category and BPEL module.

To import SAVAGE web service WSDL, simply right click SAVAGE web services and invoke “generate and copy WSDL” to BPEL module.

Drag and drop WSDL onto the drawing panel. After which, drag and drop necessary WSBPEL activities (web service, basic and structure) from the right side panel into the drawing panel to construct WSBPEL composite workflow (see Figure 61).

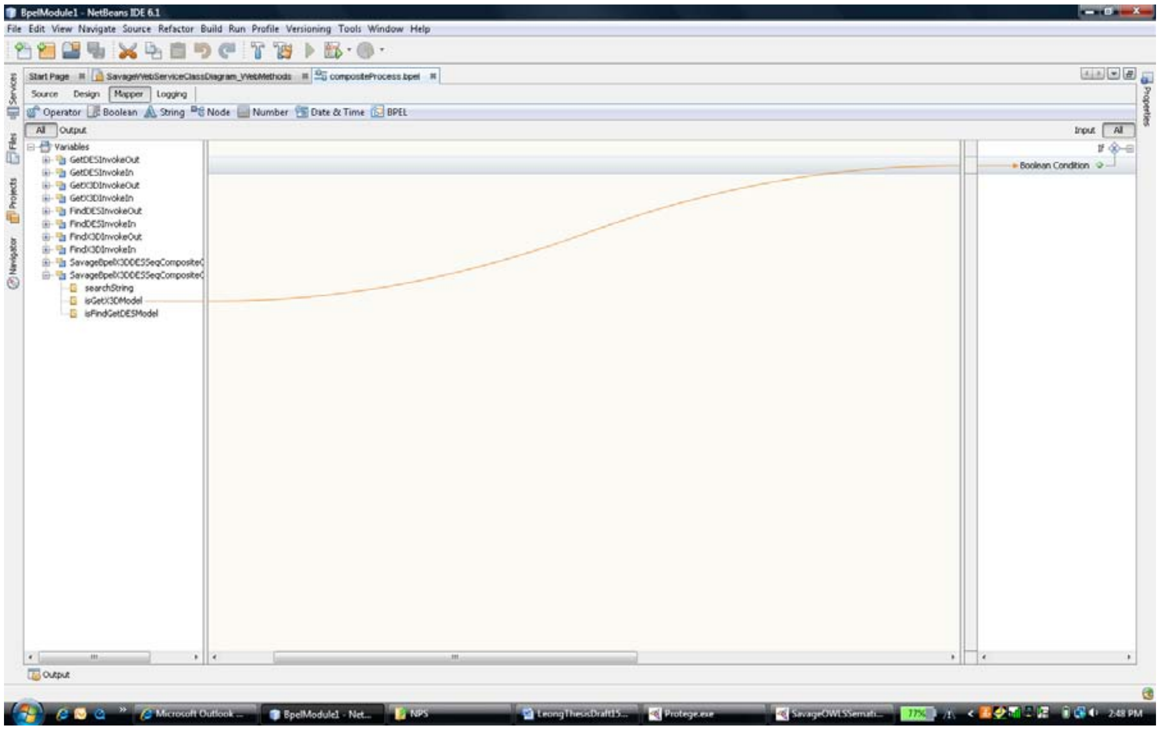

BpelModule1 - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Start Page
SavageWebServiceClassDiagram_WebMethods
compositeProcess.bpel
Source Design Mapper Logging
Operator Boolean String Node Number Date & Time BPEL
All Output
Variables
GetDESInvokeOut
GetDESInvokeIn
GetX3DInvokeOut
GetX3DInvokeIn
FindDESInvokeOut
FindDESInvokeIn
FindX3DInvokeOut
FindX3DInvokeIn
SavageBpelX3DDESSeqComposite
SavageBpelX3DDESSeqComposite
searchString
isGetX3DModel
isFindGetDESModel
Input All
Boolean Condition
Output

Figure 61. WSBPEL mapper view in NetBeans. The mapper creates WSBPEL assign activity which maps/copies the output parameter/variable of one web method to the input parameter of another web method.

In this example, the composite workflow is triggered by the *SavageBpelX3DDESSeqLink* web service. *startComposite* activity receives the invocation from *SavageBpelX3DDESSeqLink*. The input parameter to the web service is *searchString* (keyword of interest), *isGetX3DModel* (type Boolean to indicate whether to retrieve an X3D model) and *isFindGetDESModel* (type Boolean to indicate whether to find matching DES behavior URLs and retrieve their associated models).

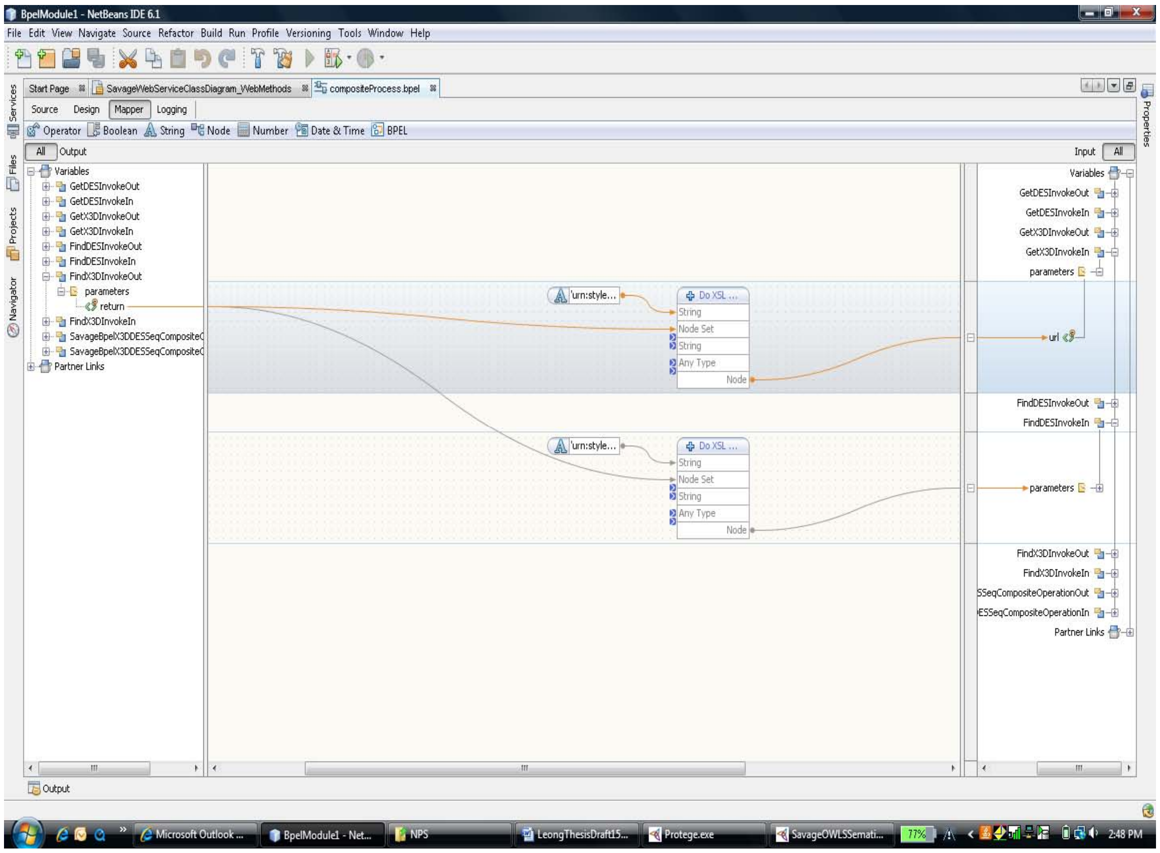

BpelModule1 - NetBeans IDE 6.1
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Start Page
SavageWebServiceClassDiagram_WebMethods
compositeProcess.bpel
Source Design Mapper Logging
Operator Boolean String Node Number Date & Time BPEL
All Output
Input All
Variables
GetDESInvokeOut
GetDESInvokeIn
GetX3DInvokeOut
GetX3DInvokeIn
FindDESInvokeOut
FindDESInvokeIn
FindX3DInvokeOut
parameters
return
FindX3DInvokeIn
SavageBpelX3DDESSeqComposite
SavageBpelX3DDESSeqComposite
Partner Links
'urn:style...
Do XSL ...
String
Node Set
String
Any Type
Node
url
parameters
SSeqCompositeOperationOut
ESSeqCompositeOperationIn
Output

Figure 62. WSBPEL NetBeans mapper view with *doXSLTransform*. *doXSLTransform* defines adaptation rules that resolve syntactic or/and semantic mismatches between two parameters.

The Assign activity maps/copies from the output parameter/variable of one web method to the input parameter of another web method. If there is a syntactic or semantic mismatch between the two parameters, *doXSLTransform* can be inserted to define the transformation rule that is required to translate between the two parameters (Figure 62). doXSLTransform will form part of the assign construct. For SAVAGE web services, *InvokeX3DFind* returns a list of X3D names and the corresponding URLs. *doXSLTransform* extracts the X3D URL from the first item of the list. The name of the style sheet is *transfromX3DURLList*. The output is compatible (syntactically and semantically glued) with the input of *InvokeDESFind* and *InvokeX3DGet*.

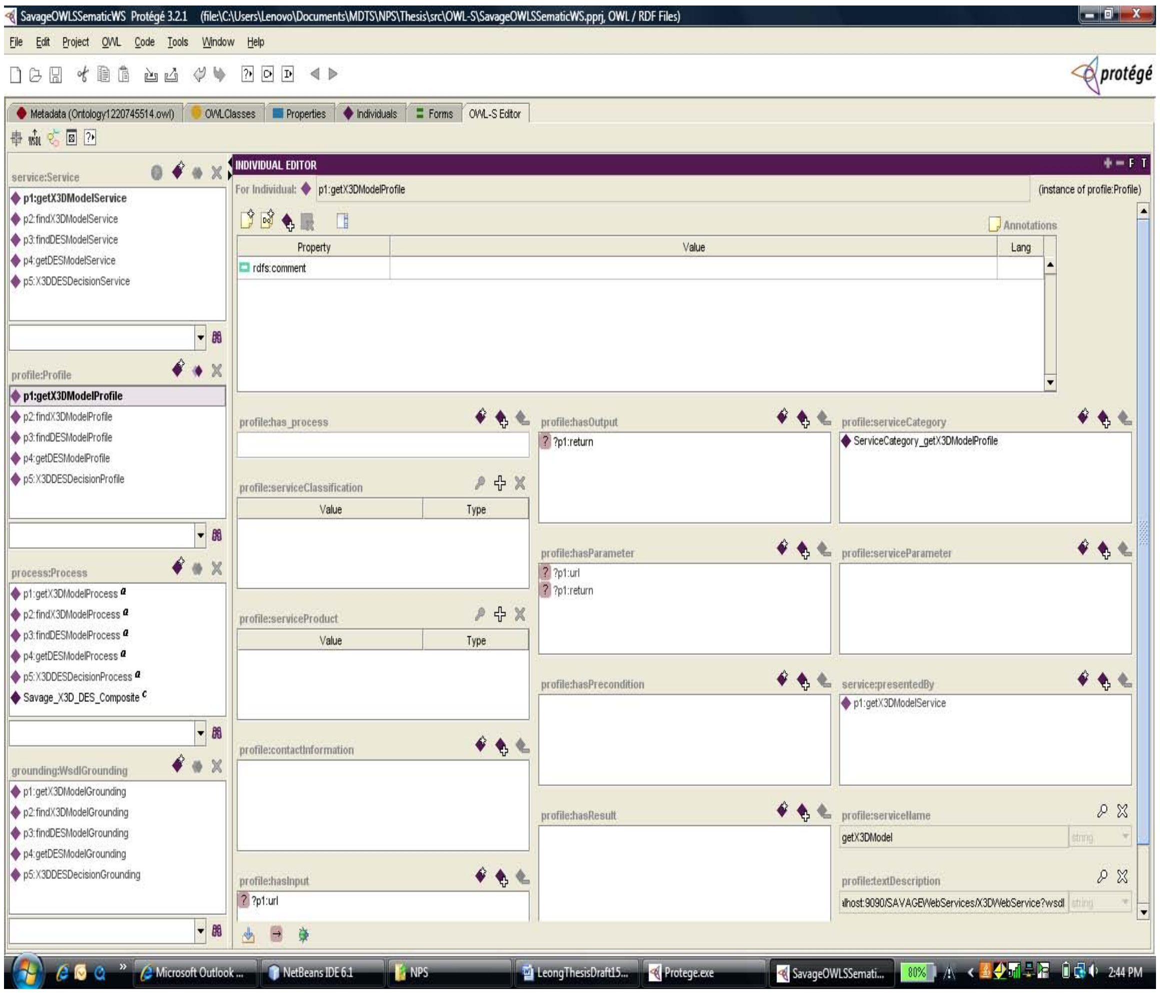
Partner link represents a partner web service. Initiate invoke, reply and receive web service activities will create a partner link with the SAVAGE web services. For example, *InvokeX3DFind* web service invoke activity creates a partner link with the *findX3DModel* web method from *X3DWebService*.

The Sequence structured activity is basically a container that organizes predefined activities and executes them in sequential order. For the SAVAGE web service composite workflow, *InvokeX3DFind* will trigger *InvokeX3DGet* or *InvokeDESFind* in sequence depending on the *if* condition. The Sequence construct is also used for *InvokeDESFind* and *invokeDESGet* when the *isFindGetDES* condition is true. If *isGetX3DModel* is true, then *InvokeX3DGet* will be triggered. If neither Boolean variable is true, the composite workflow does nothing.

The composite workflow is terminated by a web service reply, coupled with the result string to the initiated web service *SAVAGEBpelX3DDESSeqLink*.

NetBeans BPEL dynamically generates WSBPEL in XML as one configures the composite workflow in the NetBeans user interface. The result of the generated WSBPEL in XML is provided in Appendix B.2.

D. SAVAGE WS ORCHESTRATION AND ADAPTATION USING OWL-S



SavageOWLSSematicWS Protégé 3.2.1 (file:\C:\Users\Lenovo\Documents\MDTS\NPS\Thesis\src\OWL-S\SavageOWLSSematicWS.pprj, OWL / RDF Files)

service:Service
- p1:getX3DModelService
- p2:findX3DModelService
- p3:findDESModelService
- p4:getDESModelService
- p5:X3DDESDecisionService

profile:Profile
- p1:getX3DModelProfile
- p2:findX3DModelProfile
- p3:findDESModelProfile
- p4:getDESModelProfile
- p5:X3DDESDecisionProfile

process:Process
- p1:getX3DModelProcess a
- p2:findX3DModelProcess a
- p3:findDESModelProcess a
- p4:getDESModelProcess a
- p5:X3DDESDecisionProcess a
- Savage_X3D_DES_Composite c

grounding:WsdlGrounding
- p1:getX3DModelGrounding
- p2:findX3DModelGrounding
- p3:findDESModelGrounding
- p4:getDESModelGrounding
- p5:X3DDESDecisionGrounding

For Individual: p1:getX3DModelProfile (instance of profile:Profile)

Property	Value	Lang
rdfs:comment		

profile:has_process

profile:hasOutput: ?p1:return

profile:serviceCategory: ServiceCategory_getX3DModelProfile

profile:serviceClassification

Value	Type

profile:hasParameter: ?p1:url, ?p1:return

profile:serviceParameter

profile:serviceProduct

Value	Type

profile:hasPrecondition

service:presentedBy: p1:getX3DModelService

profile:contactInformation

profile:hasResult

profile:serviceName: getX3DModel string

profile:hasInput: ?p1:url

profile:textDescription: lhost:9090/SAVAGEWebServices/X3DWebService?wsdl string

Figure 63. Protégé OWL-S editor can be used to describe semantic web services in OWL-S.

The Protégé OWL-S plug-in (Figure 63) is written by SRI International. SRI is an independent, nonprofit research institute conducting client-sponsored research and development for government agencies, commercial businesses, foundations and other organizations. SRI also brings its innovation to marketplace by licensing its intellectual property and creating new ventures (http://www.sri.com/about).

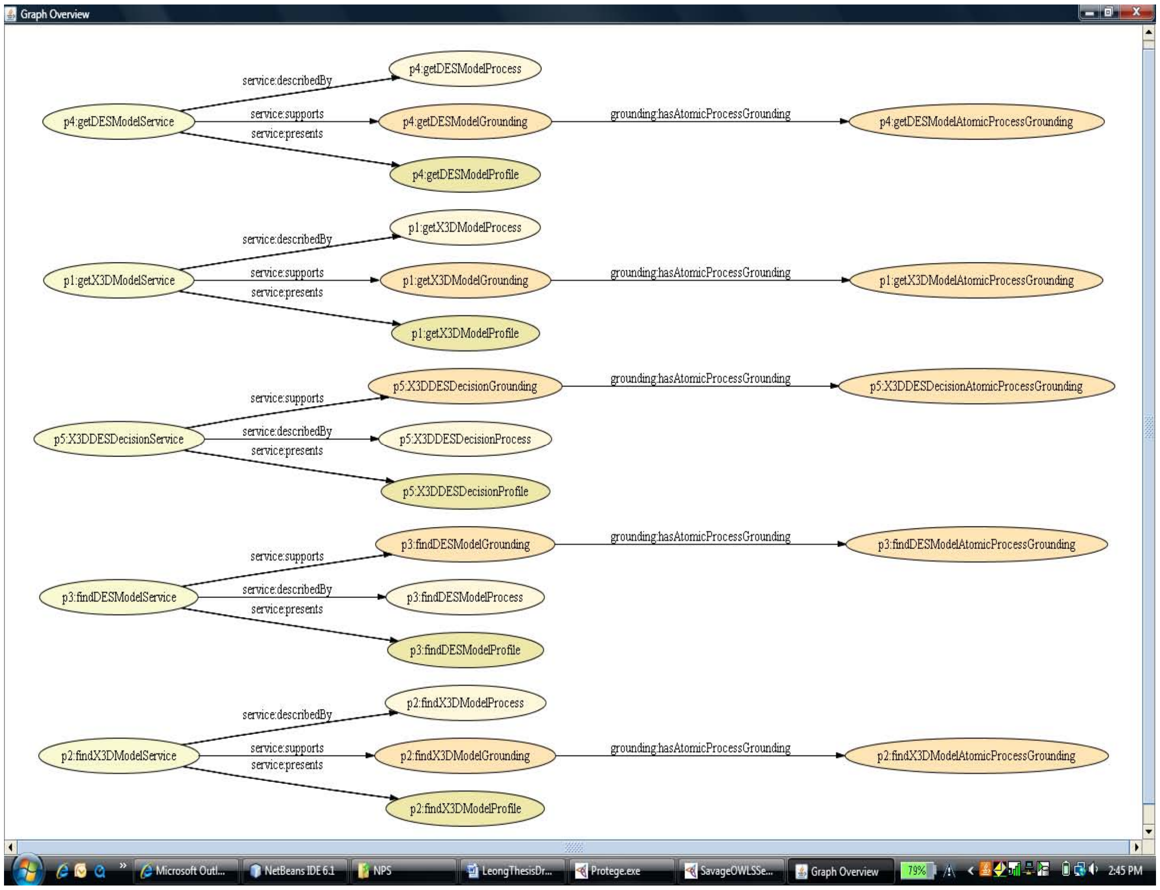


Figure 64. Graph overview of SAVAGE OWL-S service profiles, processes and groundings in Protégé.

A Wizard in the plug-in is used to import SAVAGE web services WSDL into OWL-S. OWL-S services, profiles, atomic processes for the web methods and WSDL groundings are automatically established (See Figure 64). OWL-S is very expressive. One can also add more descriptions on SAVAGE web services using the OWL-S plug-in, such as service category, free text comments, etc.

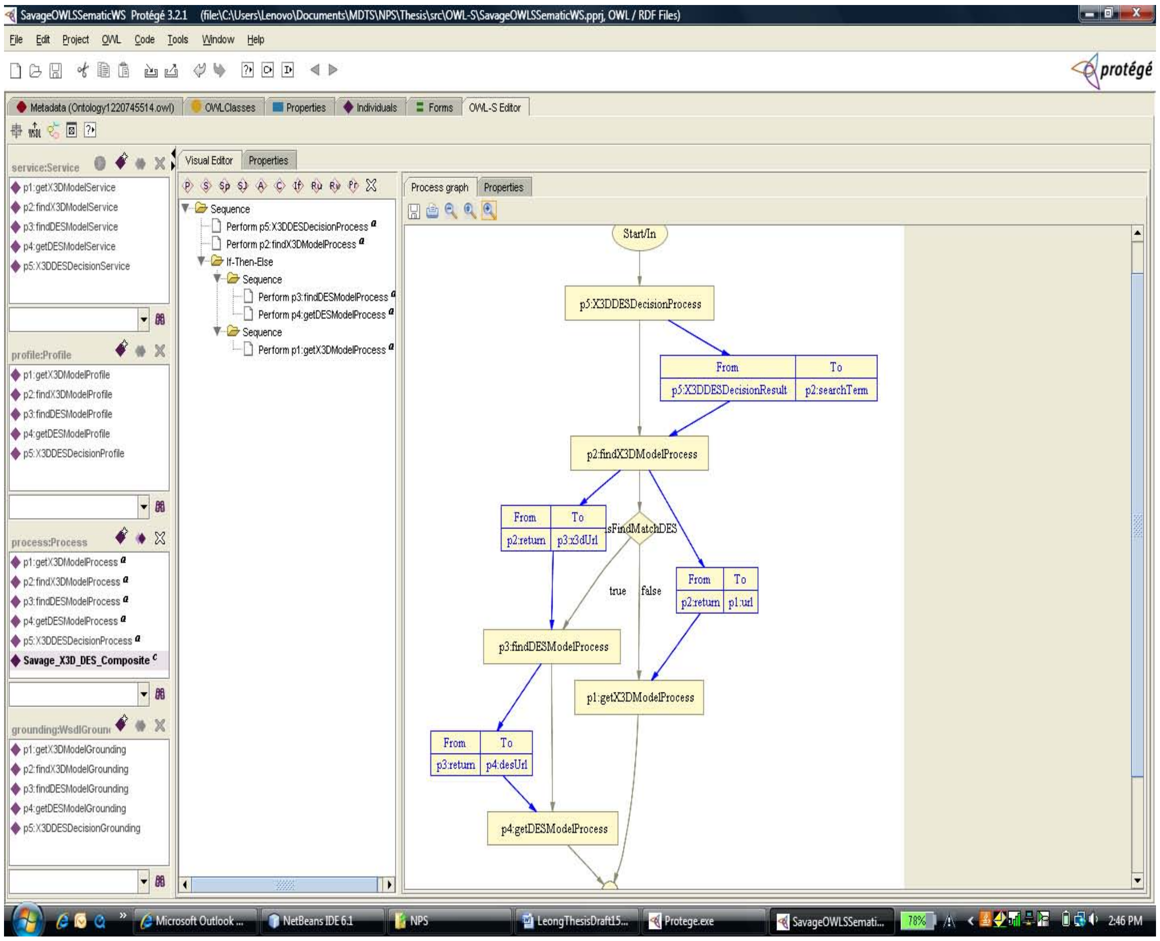


Figure 65. SAVAGE OWL-S composite process constructed using the Protégé OWL-S editor.

A similar composite workflow sequence that was done in WSBPEL is created in OWL-S (Figure 65). The composite process is a sequence. It starts off by performing *X3DDESDecisionProcess*, which has input parameters *searchTerm*, *isGetX3D* (Boolean type, to indicate whether to retrieve a X3D model from the SAVAGE repository) and *isGetMatchingDES* (Boolean type, to indicate whether to find and get the matching DES behavior model given the X3D model URL). An if-then-else construct is performed after *FindX3DModelProcess*. *FindX3DModelProcess* will invoke *findDESModelProcess* and then *getDESModelProcess* if the *isGetMatchingDES* variable is true. If *isGetX3D* variable is true, then *getX3DModelProcess* will be invoked instead. The result string will be an X3D model or DES model.

The Protégé OWL-S plug-in dynamically generates OWL-S in XML as one configures the composite workflow on the user interface. The result of the generated OWL-S in XML is in Appendix B-1.

E. COMPARISON BETWEEN WSBPEL AND OWL-S

OWL-S is based on RDF, RDFS and OWL. Hence, the language is more expressive. For example, OWL-S is able to import WSDL and the information forms part of OWL-S. OWL-S also enables specific description using comments, service category, service classification, service precondition and contact information in service profile. It also allows an instance of the service result to have additional process description such as in-condition and has-result expressions. However, the adaptation required to address the syntactic and semantic mismatches between two parameters of atomic processes is not addressed in OWL-S.

On the other hand, WSBPEL has the assign construct. When coupled with *doXSLTransform*, this is able to capture rules required to link two web services and overcome heterogeneity/mismatch problems when different services use different vocabularies. In general, WSBPEL is more established in the area of the service orchestration layer as compared to OWL-S. Hence, WSBPEL has more comprehensive basic web services and structured activities to construct composite workflows. On the other hand, WSBPEL is not as expressive as OWL-S. WSBPEL is equivalent only to the service model class of OWL-S. WSBPEL is not able to describe web services profile, WSDL grounding, etc. Hence, WSBPEL needs to work with other open standards such as WSDL and UDDI for a complete SOA solution.

F. COMPARISON BETWEEN WSMO AND OWL-S

It is reported that most of the elements/constructs described in OWL-S can be modeled in WSMO (Lara *et al.*, 2004). OWL-S is more detailed in the area of service orchestration (realized by the service model) and WSDL grounding as compared to WSMO. However, OWL-S does not have the adaption capability to resolve mismatch problems between two web methods. On the other hand, WSMO has a mediator component. Similar to the assign element in WSBPEL, the purpose of the mediator in

WSMO is to resolve syntactic and semantic mismatch problems between goals, ontologies or web services. The types of WSMO mediators are *ggMediators* (between two goals), *ooMediators* (between two ontologies), *wgMediators* (between web service and goal) and *wwMediators* (between two web services). The details of the conceptual comparison between OWL-S and WSMO are summarized in Table 2.

Comparison aspect	**WSMO-Standard**	**OWL-S**
Purpose	Focused goal, specific application domains	Wide goal, does not focus on concrete application domains
Principles	Explicit conceptual work and well-established principles	Not explicit, development based set of tasks to be solved and foundations inherited from other research areas
Coupling	Loose coupling, independent definition of description elements	Tighter coupling in several aspects
Extensibility	Extensible in every direction	Limited extensibility, mainly through OWL subclassing
Implementation and business layers	Will be clearly separated in WSMO-Full	Overlapped at some points eg. use of the Resource concept
Registry	Not dictated	Not dictated
Requester needs and service capabilities	Two different points of view, modeled independently and linked via wgMediators	Not separated, unified view in the service profile
Functionality description	Explicit and complete description	Does not describe some aspects of the functionality
Non-functional properties	Pre-defined properties. Flexible extension but not explicit mechanism	Few pre-defined properties. Explicit extension mechanism but improvable flexibility

Orchestration	Supports static and dynamic composition, but under-defined	Limited dynamic composition, completely defined
Grounding	Multiple groundings, not pre-defined grounding	Problems with multiple groundings for atomic processes, WSDL pre-defined grounding
Mediation	Scalable mediation between loosely coupled elements	No mediation
Layering	3-layers (WSMO-Lite, WSMO-Standard, WSMO-Full) covering different complexity levels of domain	No layering (layering inherited from OWL, does not reflect complexity of the application domain)
Languages	F-Logic for logical expressions. Ontology language not imposed	Language for conditions not defined. Ontology language OWL.

Table 2. Conceptual comparison between OWL-S and WSMO (From Lara *et al.*, 2004).

G. WSAIF SOFTWARE AGENTS+DATA=WEB SERVICES INTEGRATION ON THE FLY

WSAIF creates the object oriented abstraction between the component application programming interface (API) and the various SOA related open standards. The WSAIF Orchestration component integrates open standards WSBPEL, OWL-S and WSMO. It also exposes a common set of APIs that is used to interoperate with the APIs from other WSAIF components. For example, *WSAIFAbsSequence* is the abstraction to the sequencing of web services activities. *WSAIFOwlsSequence* and *WSAIFBpelSequence* extends *WSAIFAbsSequence* and implements the sequence structure representation for OWL-S and WSBPEL. WSAIF Orchestration provides generic API that receives information about the sequence activities. The user does not need to know about the details of the modeling language, which can be OWL-S, WSBPEL or both.

The WSAIF Orchestration also contains *WSAIFServiceAgentFactory* with the purpose of generating corresponding service agents for each partnering atomic web method that forms the composite process. Each service agent will acquire information on web methods such as choreography, semantics and various adaptation logics. Adaptation logics are subsumed under the WSAIF Adaption component. Service agents will work together and resolve local interoperability issues between web methods.

At the global level, the *WSAIFWorkflowAgent* will acquire knowledge about the orchestrated workflow and coordinate with the service agents. WSAIF *MatchmakeAgent* will work with service agents to explore, match services and attempt to construct possible orchestrated workflows.

Data for composite workflow structures can be captured in OWL-S, WSBPEL or WSMO. Data for adaptation logic can be captured in the assign element in WSBPEL or the mediator component in WSMO. These data are represented as XML and XML is portable. In order words, when these data are shared in the network, coupled with WSAIF software agents facilitating dynamic integration of web services into WSA, web services integration on the fly is made possible from the perspective of a new user.

Hence, WSAIF orchestration software agents will enable capabilities such as automating WS invocation, WS workflow coordination, WS workflow monitoring and service-to-service adaptation.

H. SUMMARY

This chapter described the orchestration scenario for SAVAGE web services. This was followed by the realization of the orchestration scenario using OWL-S and WSBPEL. A comparison between OWL-S and WSBPEL was made based on observation of the orchestrated composite process. Comparison of OWL-S and WSMO was summarized from open literature. The chapter concluded by elaborating on the approach to integrate WSAIF software agents and data model to enable web services integration on the fly.

VII. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

This thesis described the motivation for WSAIF as an enabler for web services integration on the fly. Its envisioned capabilities were covered in Chapter III. Chapter V to VII covered the approach taken to explore the concept for the MOVES domain. Details on the OO design of the SAVAGE web services were also discussed. The implementation setup and test results for SAVAGE web services were covered in Chapter VI. The test results were verified correct. Work showed that OWL-S, BPEL and WSMO are the possible options to model the integration, orchestration and adaptation of web services in a composite process. The comparison between the different modeling techniques was also discussed. The thesis further explained how WSAIF software agents and modeling data enable web services integration on the fly.

B. RECOMMENDATIONS FOR FUTURE WORK

Future work includes the survey, design and implementation of WSAIF components. SAVAGE web services can be extended to include more web methods, such as updating X3D and DES models. The logic for searching X3D and DES models can be further enhanced by leveraging search engines such as Lucene. COTS search engines can provide advanced search capabilities such as concept and pattern searches. SAVAGE web services can incorporate a more complex and intelligent web method. The user does need to know any specific input format or parameters. The intelligent web method takes in a simple keyword. The web service receives the invocation request, which then triggers the composite process modeled by WSBPEL or OWL-S. WSAIF facilitates the execution of the composite process in the runtime environment. The result is parsed as XML and will be returned with an associated schema. The schema will give clarity to users on the format and meaning of the results.

It is also possible that the security architecture for XML Document-Centric Security (Williams, 2008) can be incorporated into the WSAIF Security component.

Other tools and services such as a GeoServer Geographic Information System (GIS) service can be integrated and discoverable as a web service so that WSAIF is able to discover and match it with other services. Providing geographical information will certainly add value to the WSAIF-composed business processes.

It will also be interesting to study the application of various AI planning techniques on WSA and their effectiveness based on specific scenarios. The WSAIF Agent component can incorporate suitable AI planning techniques that run on SOA-related open standards. The WSAIF Agent then intelligently recommends and invokes the most suitable AI planning technique with optimal parameters set based on the identified scenario.

The Department of Defense Architecture Framework (DoDAF) defines a standard way to organize an enterprise architecture (EA) or systems architecture into complementary and consistent views. All major U.S. Government Department of Defense (DoD) programs are required to adhere to the architecture views defined in DoDAF (http://en.wikipedia.org/wiki/Department_of_Defense_Architecture_Framework). Hence, the way forward is to incorporate DoDAF architecture views for the SAVAGE web services.

Other work includes implementing the WSAIF framework to fully realize a generic, flexible, scalable, usable and intelligent web service architecture. Future work also includes the study and application of the WSAIF for global web-based simulation and visualization, driven by net-centric tactical data. The WSAIF can also be applied and tested in different domains.

APPENDIX A. SAVAGE WEB SERVICES SOURCE CODE

1. WSMETHODS CLASSES

WSMethods component contains Java classes *X3DWebService.java* and *DESWebService.java*. These classes contain web methods *findX3DModel*, *getX3DModel*, *findDESModel* and *getDESModel*. The web methods correspond to the four use cases for SAVAGE web services.

a. *X3DWebService.java*

```
package X3DWSMethod;

import WSController.X3DFindStrategy;
import WSController.X3DGetStrategy;
import WSController.WSSingleton;
import WSController.WSStrategy;
import WSModel.WSCatalogReader;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 * This class defines the webmethods for SAVAGE X3D Webservice
 * <p>
 * Currently, 2 webmethods are developed, namely :
 * findX3DModel and getX3DModel
 *
 * @author       Leong, Hoe Wai
 * @version      %I%, %G%
 * @since
 */
@WebService()
public class X3DWebService {

    /**
     * Defines Web Service Operation findX3DModel
     * The purpose of this webmethod is to find X3D URLs
     * given a search term
     *
     * @param searchTerm    search term in String
     * @return              List of X3D URLs
     * @see
     * @since
     */
    @WebMethod(operationName = "findX3DModel")
    public String findX3DModel(@WebParam(name = "searchTerm")
    String searchTerm) {
        X3DFindStrategy findStrategy =
                new X3DFindStrategy(searchTerm);
        String strategyResult =
                executeWebServiceOperation(findStrategy);
        //TODO write your implementation code here:
        return strategyResult;
    }

   /**
     * Defines executeWebServiceOperation method.
     * The method takes in an abstract WSStrategy.
     * It execute the business operation base on the specific
     * strategy passed in.
     *
     * @param WSStrategy    Abstract Strategy
     * @return          Any return XML. Content depending on the
     * specific strategy passed in
     * @see
     * @since
     */
    private String
         executeWebServiceOperation(WSStrategy specificStrategy) {
        return (specificStrategy.executeOperation());
    }

    /**
     * Defines Web Service Operation getX3DModel
     * The purpose of this webmethod is to get X3D XML
     * from Savage File Server
     *
     * @param searchTerm    search term in String
     * @return          List of X3D URLs
     * @see
     * @since
     */
    @WebMethod(operationName = "getX3DModel")
    public String getX3DModel(@WebParam(name = "url")
    String url) {
        //TODO write your implementation code here:
        X3DGetStrategy getStrategy = new X3DGetStrategy(url);
        String strategyResult =
                executeWebServiceOperation(getStrategy);
        return strategyResult;
    }
}
```

b. DESWebService.java

```
package DESWSMethod;

import WSController.DESFindStrategy;
import WSController.DESGetStrategy;
import WSController.WSStrategy;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
```

```
 * This class defines the webmethods for SAVAGE DES Webservice
 * <p>
 * Currently, 2 webmethods are developed, namely :
 * findDESModel and getDESModel
 *
 * @author      Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */

@WebService()
public class DESWebService {

    /**
     * Defines Web Service Operation findDESModel
     * The purpose of this webmethod is to find associated DES
     * behavior given X3D URL. The information is found in
     * SavageCatalog, within SMAL
     *
     * @param x3dUrl    The X3D URL
     * @return          List of associated DES URLs
     * @see
     * @since
     */
    @WebMethod(operationName = "findDESModel")
    public String findDESModel(@WebParam(name = "x3dUrl")
    String x3dUrl) {
        //TODO write your implementation code here:
        DESFindStrategy findStrategy = new DESFindStrategy(x3dUrl);
        String strategyResult =
                executeWebServiceOperation(findStrategy);
        return strategyResult;
    }

    /**
     * Defines Web Service Operation getDESModel
     * The purpose of this webmethod is to get DES XML
     * given DES URL. The information is found in Savage file server
     *
     * @param desUrl    The DES URL
     * @return          DES XML
     * @see
     * @since
     */
    @WebMethod(operationName = "getDESModel")
    public String getDESModel(@WebParam(name = "desUrl")
    String desUrl) {
        //TODO write your implementation code here:
        DESGetStrategy getStrategy = new DESGetStrategy(desUrl);
        String strategyResult =
                executeWebServiceOperation(getStrategy);
        return strategyResult;
    }

    /**
     * Defines executeWebServiceOperation method.
     * The method takes in an abstract WSStrategy.
     * It execute the business operation base on the specific
     * strategy passed in.
     *
     * @param WSStrategy    Abstract Strategy
     * @return          Any return XML. Content depending on the
     * specific strategy passed in
     * @see
     * @since
     */
    private String
            executeWebServiceOperation(WSStrategy specificStrategy) {
        return (specificStrategy.executeOperation());
    }
}
```

2. WSCONTROLLER CLASSES

WSController component contains Java classes that realize the strategy for each web method. WSSingleton.java creates and maintains the global static/single instance of a particular class or model. WSUtility.java basically contains reusable utility operations such as getContentFromFile.

a. WSStrategy.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package WSController;

/**
 * This is an abstract WSStrategy class
 *
 * @author      Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */
public abstract class WSStrategy {
    /**
     * Abstract method for executing web service strategy
     *
     * @param
     * @return String     Return XML depending on the specific
     * strategy executed
     * @see
     * @since
     */
        public abstract String executeOperation();
}
```

b. X3DFindStrategy.java

```
package WSController;

import WSModel.X3DFindResultEntity;
import java.io.StringWriter;
import java.util.HashMap;
import java.util.StringTokenizer;
import org.exolab.castor.xml.Marshaller;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * This class implements find X3D strategy
 *
 * @author      Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */
public class X3DFindStrategy extends WSStrategy {
    private String inputStr;
    private String outputStr;
    private HashMap<String,String> findResultSet;

    public X3DFindStrategy(String input) {
       this.setInputStr(input);
       findResultSet = new HashMap<String,String>();
    }

    public String getInputStr() {
        return inputStr;
    }

    public void setInputStr(String inputStr) {
        this.inputStr = inputStr;
    }

    public String getOutputStr() {
        return outputStr;
    }

    public void setOutputStr(String outputStr) {
        this.outputStr = outputStr;
    }

    public HashMap<String,String> getFindResultSet() {
        return findResultSet;
    }

    public void
       setFindResultSet(HashMap<String,String> findResultSet) {
        this.findResultSet = findResultSet;
    }

    /**
     * This method execute find strategy for X3D.
     * The method first get catalogDocument from WSSingleton
     * Then iterate through DOM to find matching search term
     * in SavageCatalog
     *
     * @param
     * @return String         List of X3D URLs in XML
     * @see
     * @since
     */
    public String executeOperation() {
        System.out.println("X3DFindStrategy : start executing " +
                "X3DFindStrategy ... ");
        Document catalogDocument =WSSingleton.getCatalogDocument();
        iterateChild(catalogDocument);
        X3DFindResultEntity resultEntity =
                new X3DFindResultEntity(this.getFindResultSet());
        StringWriter strWrite = new StringWriter();
        try {
            Marshaller marshaller = new Marshaller(strWrite);
            marshaller.marshal(resultEntity);
        } catch (Exception ex) {
            System.out.println("Exception : "+ex);
        }
        System.out.println("X3DFindStrategy : end executing " +
                "X3DFindStrategy ... ");
        return strWrite.toString();
    }

    /**
     * This method iterates DOM
     *
     * @param Node      specific node of DOM
     * @return
     * @see
     * @since
     */
    private void iterateChild(Node node) {
        String nodeName;
        NodeList childList = node.getChildNodes();
        int childListLength = childList.getLength();
        for (int i=0; i<childListLength; i++) {
            Node currentNode = childList.item(i);
            nodeName = currentNode.getNodeName();

            if (nodeName.compareTo("Page")==0)
                searchPage(currentNode);

            iterateChild(currentNode);
        }
    }

    /**
     * This method search attribute url, name, description
     *
     * @param Node      specific page node of DOM
     * @return
     * @see
     * @since
     */
    private void searchPage(Node currentNode) {
        boolean isFound = false;
        String urlTemp="";
        String nameTemp="";
        NamedNodeMap attrs = currentNode.getAttributes();
        int len = attrs.getLength();
        for (int j=0; j<len; j++) {
            Attr attr = (Attr)attrs.item(j);

            if (attr.getNodeName().compareTo("url")==0)
                urlTemp = attr.getNodeValue();

            if (attr.getNodeName().compareTo("name")==0)
                nameTemp = attr.getNodeValue();


            if ((attr.getNodeName().compareTo("name")==0) &&
                    (attr.getNodeValue().compareTo(inputStr)==0))
                isFound=true;

            if ((attr.getNodeName().compareTo("title")==0) &&
                    (attr.getNodeValue().compareTo(inputStr)==0))
                isFound=true;

            if (attr.getNodeName().compareTo("description")==0)
                if (searchDescription(attr.getNodeValue()))
                    isFound=true;
        }

        if (isFound)
            this.getFindResultSet().put(nameTemp, urlTemp);
    }

    /**
     * This method search string tokenize description and do
     * keyword match with search term
     *
     * @param String      description
     * @return   <code>true</code> if there is keyword match
     * between search term and description
     *           <code>false</code> if there are no matches found.
     * @see
     * @since
     */
    private boolean searchDescription(String description) {
        // code logic for searching description...
        boolean isSearchDesc=false;
        StringTokenizer descTokens =
                new StringTokenizer(description," ");

        while (descTokens.hasMoreTokens())
            if (descTokens.nextToken().compareTo(inputStr)==0)
                isSearchDesc=true;

        return isSearchDesc;
    }
}
```

c. X3DGetStrategy.java

```
package WSController;

import java.util.StringTokenizer;

/**
 * This class implements get X3D strategy
 *
 * @author       Leong, Hoe Wai
 * @version      %I%, %G%
 * @since
 */
public class X3DGetStrategy extends WSStrategy {

    private String inputStr;

    public X3DGetStrategy(String input) {
       this.setInputStr(input);
    }

    public String getInputStr() {
        return inputStr;
    }

    public void setInputStr(String inputStr) {
        this.inputStr = inputStr;
    }

    /**
     * This method execute get strategy for X3D.
     * The method finds common directory "Savage"
     * and retrive X3D XML from file server base on the file path
     *
     * @param
     * @return String         X3D XML
     * @see
     * @since
     */
    public String executeOperation() {
        System.out.println("X3DGetStrategy : start executing " +
                "X3DGetStrategy ... ");
        String beginFilePath = WSSingleton.getSavagePath();
        StringTokenizer token =
                new StringTokenizer(this.getInputStr(),"/");

        while (token.hasMoreTokens())
            if (token.nextToken().compareTo("Savage")==0)
```

```
                break;

        String remainFilePath = "";
        while (token.hasMoreTokens())
            remainFilePath=
                    remainFilePath.concat("/"+token.nextToken());

        String fileLocation = beginFilePath+remainFilePath;
        String xmlStr = WSUtility.getContentsFromFile(fileLocation);

        System.out.println("X3DGetStrategy : end executing " +
                "X3DGetStrategy ... ");
        return xmlStr;
    }
}
```

d. DESFindStrategy.java

```
package WSController;

import WSModel.DESFindResultEntity;
import java.io.StringWriter;
import java.util.HashMap;
import org.exolab.castor.xml.Marshaller;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * This class implements find DES strategy
 *
 * @author      Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */
public class DESFindStrategy extends WSStrategy {
    private String inputStr;
    private String outputStr;
    private HashMap<String,String> findResultSet;

    public DESFindStrategy(String input) {
        this.setInputStr(input);
        findResultSet = new HashMap<String,String>();
    }

    public String getInputStr() {
        return inputStr;
    }

    public void setInputStr(String inputStr) {
        this.inputStr = inputStr;
    }

    public String getOutputStr() {
        return outputStr;
    }

    public void setOutputStr(String outputStr) {
        this.outputStr = outputStr;
    }

    public HashMap<String, String> getFindResultSet() {
        return findResultSet;
    }

    public void setFindResultSet
            (HashMap<String, String> findResultSet) {
        this.findResultSet = findResultSet;
    }

    /**
     * This method execute find strategy for DES.
     * The method first get catalogDocument from WSSingleton
     * Then iterate through DOM to find matching X3D url in
     * SavageCatalog
     *
     * @param
     * @return String   List of DES URLs in XML
     * @see
     * @since
     */
    public String executeOperation() {
        System.out.println("DESFindStrategy : start executing " +
                "DESFindStrategy ... ");
        Document catalogDocument =WSSingleton.getCatalogDocument();
        iterateChild(catalogDocument);
        DESFindResultEntity resultEntity =
                new DESFindResultEntity(this.getFindResultSet());
        StringWriter strWrite = new StringWriter();
        try {
            Marshaller marshaller = new Marshaller(strWrite);
            marshaller.marshal(resultEntity);
        } catch (Exception ex) {
            System.out.println("Exception : "+ex);
        }
        return strWrite.toString();
    }

    /**
     * This method iterates DOM
     *
     * @param Node      specific node of DOM
     * @return
     * @see
     * @since
     */
    private void iterateChild(Node node) {
        String nodeName;
        NodeList childList = node.getChildNodes();
        int childListLength = childList.getLength();
        for (int i=0; i<childListLength; i++) {
            Node currentNode = childList.item(i);
            nodeName = currentNode.getNodeName();

            if (nodeName.compareTo("Page")==0)
                if (isX3D(currentNode))
                    extractBehavior(currentNode);

            iterateChild(currentNode);
        }
    }

   /**
     * This method compares node url attr with url input from
     * webmethod to see if they are the same
     *
     * @param Node      specific node of DOM
     * @return <code>true</code> if node url attr is the same as
     * url input <code>false</code> if the 2 urls differ
     * @see
     * @since
     */
    private boolean isX3D(Node node) {
        boolean isFound=false;
        NamedNodeMap attrs = node.getAttributes();
        int len = attrs.getLength();
        for (int j=0; j<len; j++) {
            Attr attr = (Attr)attrs.item(j);
            if (attr.getNodeName().compareTo("url")==0)
                if (attr.getNodeValue().compareTo(inputStr)==0)
                    isFound=true;
        }
        return isFound;
    }

    /**
     * This method extract agent name and DES url.
     *
     * @param Node     specific node in DOM
     * @return
     * @see
     * @since
     */
    private void extractBehavior(Node node) {
        String nodeName="";
        String agentTemp="";
        NodeList childList = node.getChildNodes();
        int childListLength = childList.getLength();
        for (int i=0; i<childListLength; i++) {
            Node currentNode = childList.item(i);
            nodeName = currentNode.getNodeName();
            if (nodeName.compareTo("SimulationAgent")==0) {
                NamedNodeMap attrs = currentNode.getAttributes();
                int len = attrs.getLength();
                for (int j=0; j<len; j++) {
                    Attr attr = (Attr)attrs.item(j);
                    if (attr.getNodeName().compareTo("agent")==0)
                        agentTemp=attr.getNodeValue();
                    if (attr.getNodeName().compareTo("url")==0)
                        this.getFindResultSet().put(agentTemp,
                                attr.getNodeValue());
                }
            }
            extractBehavior(currentNode);
        }
    }
}
```

e. ***DESGetStrategy.java***

```
package WSController;

import java.util.StringTokenizer;

/**
 * This class implements get DES strategy
 *
 * @author       Leong, Hoe Wai
 * @version      %I%, %G%
 * @since
 */
public class DESGetStrategy extends WSStrategy {
    private String inputStr;

    public DESGetStrategy(String input) {
       this.setInputStr(input);
    }

    public String getInputStr() {
        return inputStr;
    }

    public void setInputStr(String inputStr) {
        this.inputStr = inputStr;
    }

    /**
     * This method execute get strategy for DES.
     * The method finds common directory "ViskitModels"
     * and retrive DES XML from file server base on the file path
     *
     * @param
     * @return String      DES XML
     * @see
     * @since
     */
    public String executeOperation() {
        System.out.println("DESGetStrategy : start executing " +
                "DESGetStrategy ... ");
        String beginFilePath = WSSingleton.getViskitModelPath();
        StringTokenizer token =
                new StringTokenizer(this.getInputStr(),"/");
```

```java

        while (token.hasMoreTokens())
            if (token.nextToken().compareTo("ViskitModels")==0)
                break;

        String remainFilePath = "";
        while (token.hasMoreTokens())
            remainFilePath=
                    remainFilePath.concat("/"+token.nextToken());

        String fileLocation = beginFilePath+remainFilePath;
        String xmlStr = WSUtility.getContentsFromFile(fileLocation);

        System.out.println("DESGetStrategy : end executing " +
                "DESGetStrategy ... ");
        return xmlStr;
    }
}
```

f. *WSSingleton.java*

```java
package WSController;

import WSModel.WSCatalogReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import org.w3c.dom.Document;

/**
 * This class implements Singleton
 * holding static instances of catalogDocument
 * and static reference to catalogFilePath,catalogFileName,
 * savagePath, viskitModelPath information
 *
 * @author       Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */
public class WSSingleton {
   private static Document catalogDocument;
   private static String  PROPERTIES_FILE_NAME =
           "SavageWebServices.properties";
   private static Properties configuration;

    public WSSingleton() {

    }

    public static String getCatalogFilePath() {
        if (configuration==null) {
            InputStream in = WSSingleton.class.getResourceAsStream
                    (PROPERTIES_FILE_NAME);
            configuration = new Properties();
            try {
                configuration.load(in);
                } catch (IOException e) {
                    System.out.println("Exception at " +
                            "getCatalogFilePath : "+e);
                }
        }
        return (String)configuration.get("catalogFilePath");
    }

    public static Document getCatalogDocument() {
        if (catalogDocument==null) {
            WSCatalogReader catalogReader = new WSCatalogReader
                    (WSSingleton.getCatalogFilePath()+
                    WSSingleton.getCatalogFileName());
            catalogReader.savageCatalogConstruct();
            WSSingleton.setCatalogDocument
                    (catalogReader.getDocument());
        }
        return catalogDocument;
    }

    public static void setCatalogDocument
            (Document catalogDocument) {
        WSSingleton.catalogDocument = catalogDocument;
    }

    public static String getCatalogFileName() {
        if (configuration==null) {
            InputStream in = WSSingleton.class.getResourceAsStream
                    (PROPERTIES_FILE_NAME);
            configuration = new Properties();
            try {
                configuration.load(in);
                } catch (IOException e) {
                    System.out.println("Exception at " +
                            "getCatalogFileName : "+e);
                }
        }
        return (String)configuration.get("catalogFileName");
    }

    public static String getSavagePath() {
        if (configuration==null) {
            InputStream in = WSSingleton.class.getResourceAsStream
                    (PROPERTIES_FILE_NAME);
            configuration = new Properties();
            try {
                configuration.load(in);
                } catch (IOException e) {
                    System.out.println("Exception at " +
                            "getSavagePath : "+e);
                }
        }
        return (String)configuration.get("savagePath");
    }

    public static String getViskitModelPath() {
        if (configuration==null) {
            InputStream in = WSSingleton.class.getResourceAsStream
                    (PROPERTIES_FILE_NAME);
            configuration = new Properties();
            try {
                configuration.load(in);
            } catch (IOException e) {
                System.out.println("Exception at " +
                        "getViskitModelPath : "+e);
            }
        }
        return (String)configuration.get("viskitModelPath");
    }
}
```

g. WSUtility.java

```
package WSController;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

/**
 * WS Utility class
 *
 * @author      Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */
public class WSUtility {

  /**
  * Fetch the entire contents of a text file, and return it
  * in a String. This style of implementation does not throw
  * Exceptions to the caller.
  *
  * @param fileStr is a file which already exists and can be read
  * @return String     Return content of file
  * @see
  * @since
  */
  public static String getContentsFromFile(String fileStr) {
    StringBuilder contents = new StringBuilder();
    File aFile = new File(fileStr);
    try {
      BufferedReader input =  new BufferedReader
              (new FileReader(aFile));
      try {
        String line = null;
        while (( line = input.readLine()) != null){
          contents.append(line);
          contents.append(System.getProperty("line.separator"));
        }
      }
      finally {
        input.close();
      }
    }
    catch (IOException ex){
      ex.printStackTrace();
    }

    return contents.toString();
  }

}
```

3. WSMODEL CLASSES

WSModel component contains Java classes that read and construct the SAVAGE catalog in DOM. The instances of X3DFindResultEntity.java and DESFindResultEntity.java are used to encapsulate web methods return results before they are parsed as XML.

a. WSCatalogReader.java

```
package WSModel;


import java.io.File;
import java.io.FileNotFoundException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;

/**
 * Savage Catalog reader
 *
 * @author      Leong, Hoe Wai
 * @version     %I%, %G%
 * @since
 */
public class WSCatalogReader {
    private Document document;
    private String catalogFileName;

    public WSCatalogReader(String catalogFile)
    {
        this.setCatalogFileName(catalogFile);
    }

    /**
     * This method read SavageCatalog file and construct DOM
     *
     * @param
     * @return
     * @see
     * @since
     */
    public void savageCatalogConstruct()
    {
        System.out.println("X3DCatalogBuilder : Savage Catalog " +
                "construct started ... ");
        try
        {
            DocumentBuilderFactory factory =
                    DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();

            if (builder.isNamespaceAware()) {
                System.out.println("X3DCatalogBuilder is " +
                        "namespace aware");
            } else {
                System.out.println("X3DCatalogBuilder is not " +
                        "namespace aware");
            }

            if (builder.isValidating()) {
                System.out.println("X3DCatalogBuilder is " +
                        "validation capable");
            } else {
                System.out.println("X3DCatalogBuilder is not " +
                        "validation capable");
            }

            this.setDocument(builder.parse(new File
                    (this.getCatalogFileName())));

        } catch (ParserConfigurationException pce) {
            System.out.println("ParserConfigurationException " +
                    "occured:"+pce);
        } catch (FactoryConfigurationError fce) {
            System.out.println("FactoryConfigurationError " +
                    "occured:"+fce);
        } catch (FileNotFoundException fnfe) {
            System.out.println("FileNotFoundException occured:"
                    +fnfe);
        } catch (Exception ex) {
            System.out.println("Other exception occured: "+ex);
        }
        System.out.println("X3DCatalogBuilder : Savage Catalog " +
                "construct completed ... ");
    }

    public String getCatalogFileName() {
        return catalogFileName;
    }

    public void setCatalogFileName(String catalogFileName) {
        this.catalogFileName = catalogFileName;
    }

    public Document getDocument() {
        return document;
    }

    public void setDocument(Document document) {
        this.document = document;
    }

}
```

b. X3DFindResultEntity.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package WSModel;

import java.util.HashMap;

/**
 *
 * @author Leong, Hoe Wai
 */
public class X3DFindResultEntity {
    private HashMap<String,String> resultSet;

    public X3DFindResultEntity(HashMap<String,String> map) {
        this.setResultSet(map);
    }

    public HashMap<String,String> getResultSet() {
        return resultSet;
    }

    public void setResultSet(HashMap<String,String> resultSet) {
        this.resultSet = resultSet;
    }
}
```

c. DESFindResultEntity.java

```
package WSModel;

import java.util.HashMap;

/**
 * Entity class for DESFindStrategy result list of DES agents
 * and urls
 *
 * @author       Leong, Hoe Wai
 * @version      %I%, %G%
 * @since
 */
public class DESFindResultEntity {
    private HashMap<String,String> resultSet;

    public DESFindResultEntity(HashMap<String,String> map) {
        this.setResultSet(map);
    }

    public HashMap<String,String> getResultSet() {
        return resultSet;
    }

    public void setResultSet
            (HashMap<String,String> resultSet) {
        this.resultSet = resultSet;
    }
}
```

d. *Savage Catalog*

Savage Catalog URL is found in
https://savage.nps.edu/Savage/ContentCatalogSavage.xml

4. GENERATED RESOURCES

Generated resources are Java classes, WSDL and schemas auto-generated by JAX-WS. The generated Java classes correspond to the four web methods. JAXB is used to parse the Java classes into WSDL and associated schemas.

a. *FindX3DModel.java*

```

package X3DWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "findX3DModel", namespace =
"http://X3DWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "findX3DModel", namespace =
"http://X3DWSMethod/")
public class FindX3DModel {

    @XmlElement(name = "searchTerm", namespace = "")
    private String searchTerm;

    /**
     *
     * @return
     *     returns String
     */
    public String getSearchTerm() {
        return this.searchTerm;
    }

    /**
     * 
     * @param searchTerm
     *     the value for the searchTerm property
     */
    public void setSearchTerm(String searchTerm) {
        this.searchTerm = searchTerm;
    }

}
```

b. FindX3DModelResponse.java

```

package X3DWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "findX3DModelResponse",
namespace = "http://X3DWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "findX3DModelResponse",
namespace = "http://X3DWSMethod/")
public class FindX3DModelResponse {

    @XmlElement(name = "return", namespace = "")
    private String _return;

    /**
     * 
     * @return
     *     returns String
     */
    public String getReturn() {
        return this._return;
    }

    /**
     * 
     * @param _return
     *     the value for the _return property
     */
    public void setReturn(String _return) {
        this._return = _return;
    }

}
```

c. *GetX3DModel.java*

```

package X3DWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getX3DModel",
namespace = "http://X3DWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getX3DModel",
namespace = "http://X3DWSMethod/")
public class GetX3DModel {

    @XmlElement(name = "url", namespace = "")
    private String url;

    /**
     *
     * @return
     *     returns String
     */
    public String getUrl() {
        return this.url;
    }

    /**
     *
     * @param url
     *     the value for the url property
     */
    public void setUrl(String url) {
        this.url = url;
    }

}
```

d. *GetX3DModelResponse.java*

```

package X3DWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getX3DModelResponse", namespace =
"http://X3DWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getX3DModelResponse", namespace =
"http://X3DWSMethod/")
public class GetX3DModelResponse {

    @XmlElement(name = "return", namespace = "")
    private String _return;

    /**
     *
     * @return
     *     returns String
     */
    public String getReturn() {
        return this._return;
    }

    /**
     *
     * @param _return
     *     the value for the _return property
     */
    public void setReturn(String _return) {
        this._return = _return;
    }

}
```

e. *FindDESModel.java*

```

package DESWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "findDESModel", namespace =
                                    "http://DESWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "findDESModel", namespace =
                                    "http://DESWSMethod/")
public class FindDESModel {

    @XmlElement(name = "x3dUrl", namespace = "")
    private String x3DUrl;

    /**
     *
     * @return
     *     returns String
     */
    public String getX3DUrl() {
        return this.x3DUrl;
    }

    /**
     *
     * @param x3DUrl
     *     the value for the x3DUrl property
     */
    public void setX3DUrl(String x3DUrl) {
        this.x3DUrl = x3DUrl;
    }

}
```

f. FindDESModelResponse.java

```

package DESWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "findDESModelResponse", namespace =
                                                 "http://DESWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "findDESModelResponse", namespace =
                                                 "http://DESWSMethod/")
public class FindDESModelResponse {

    @XmlElement(name = "return", namespace = "")
    private String _return;

    /**
     *
     * @return
     *     returns String
     */
    public String getReturn() {
        return this._return;
    }

    /**
     *
     * @param _return
     *     the value for the _return property
     */
    public void setReturn(String _return) {
        this._return = _return;
    }

}
```

g. GetDESModel.java

```java

package DESWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getDESModel", namespace =
                                        "http://DESWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getDESModel", namespace =
                                        "http://DESWSMethod/")
public class GetDESModel {

    @XmlElement(name = "desUrl", namespace = "")
    private String desUrl;

    /**
     *
     * @return
     *     returns String
     */
    public String getDesUrl() {
        return this.desUrl;
    }

    /**
     *
     * @param desUrl
     *     the value for the desUrl property
     */
    public void setDesUrl(String desUrl) {
        this.desUrl = desUrl;
    }

}
```

h. GetDESModelResponse.java

```java

package DESWSMethod.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getDESModelResponse", namespace =
                                        "http://DESWSMethod/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getDESModelResponse", namespace =
                                         "http://DESWSMethod/")
public class GetDESModelResponse {

    @XmlElement(name = "return", namespace = "")
    private String _return;

    /**
     * 
     * @return
     *     returns String
     */
    public String getReturn() {
        return this._return;
    }

    /**
     * 
     * @param _return
     *     the value for the _return property
     */
    public void setReturn(String _return) {
        this._return = _return;
    }

}
```

i. *X3DWebServiceService.wsdl*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net.
    RI's version is JAX-WS RI 2.1.2-b05-RC1. -->
<definitions targetNamespace="http://X3DWSMethod/"
    name="X3DWebServiceService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://X3DWSMethod/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://X3DWSMethod/"
        schemaLocation="X3DWebServiceService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="findX3DModel">
    <part name="parameters" element="tns:findX3DModel"/>
  </message>
  <message name="findX3DModelResponse">
    <part name="parameters" element="tns:findX3DModelResponse"/>
  </message>
  <message name="getX3DModel">
    <part name="parameters" element="tns:getX3DModel"/>
  </message>
  <message name="getX3DModelResponse">
    <part name="parameters" element="tns:getX3DModelResponse"/>
  </message>
  <portType name="X3DWebService">
```

```
    <operation name="findX3DModel">
      <input message="tns:findX3DModel"/>
      <output message="tns:findX3DModelResponse"/>
    </operation>
    <operation name="getX3DModel">
      <input message="tns:getX3DModel"/>
      <output message="tns:getX3DModelResponse"/>
    </operation>
  </portType>
  <binding name="X3DWebServicePortBinding" type="tns:X3DWebService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
    <operation name="findX3DModel">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="getX3DModel">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="X3DWebServiceService">
    <port name="X3DWebServicePort"
        binding="tns:X3DWebServicePortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

j. X3DWebServiceService_schema1.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://X3DWSMethod/"
    xmlns:tns="http://X3DWSMethod/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="findX3DModel" type="tns:findX3DModel"/>

  <xs:element name="findX3DModelResponse"
    type="tns:findX3DModelResponse"/>

  <xs:element name="getX3DModel" type="tns:getX3DModel"/>

  <xs:element name="getX3DModelResponse"
    type="tns:getX3DModelResponse"/>

  <xs:complexType name="getX3DModel">
    <xs:sequence>
      <xs:element name="url" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getX3DModelResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="findX3DModel">
    <xs:sequence>
      <xs:element name="searchTerm" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="findX3DModelResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

k. DESWebServiceService.wsdl

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net.
                    RI's version is JAX-WS RI 2.1.2-b05-RC1. -->
<definitions targetNamespace="http://DESWSMethod/"
    name="DESWebServiceService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://DESWSMethod/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://DESWSMethod/"
        schemaLocation="DESWebServiceService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="findDESModel">
    <part name="parameters" element="tns:findDESModel"/>
  </message>
  <message name="findDESModelResponse">
    <part name="parameters" element="tns:findDESModelResponse"/>
  </message>
  <message name="getDESModel">
    <part name="parameters" element="tns:getDESModel"/>
  </message>
  <message name="getDESModelResponse">
    <part name="parameters" element="tns:getDESModelResponse"/>
  </message>
  <portType name="DESWebService">
    <operation name="findDESModel">
```

```
      <input message="tns:findDESModel"/>
      <output message="tns:findDESModelResponse"/>
    </operation>
    <operation name="getDESModel">
      <input message="tns:getDESModel"/>
      <output message="tns:getDESModelResponse"/>
    </operation>
  </portType>
  <binding name="DESWebServicePortBinding" type="tns:DESWebService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document"/>
    <operation name="findDESModel">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="getDESModel">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="DESWebServiceService">
    <port name="DESWebServicePort"
        binding="tns:DESWebServicePortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

l. DESWebServiceService_schema1.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://DESWSMethod/"
    xmlns:tns="http://DESWSMethod/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="findDESModel" type="tns:findDESModel"/>

  <xs:element name="findDESModelResponse"
    type="tns:findDESModelResponse"/>

  <xs:element name="getDESModel" type="tns:getDESModel"/>

  <xs:element name="getDESModelResponse"
    type="tns:getDESModelResponse"/>

  <xs:complexType name="getDESModel">
```

```
    <xs:sequence>
      <xs:element name="desUrl" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getDESModelResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="findDESModel">
    <xs:sequence>
      <xs:element name="x3dUrl" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="findDESModelResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

5. JSP CLIENT

JSP client is developed to demonstrate client invocation of SAVAGE web services (aka use cases). It is also used to test and verify that the results returned by SAVAGE web services are correct.

a. index.jsp

```
<%--
  Document   : index
  Created on : Oct 13, 2008, 10:22:14 PM
  Author     : Leong, Hoe Wai
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
        charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h2>Welcome to SAVAGE Web Services Test Page</h2>

        <form action="SavageWSClientServlet.jsp">
        <input type="radio" name="webmethod" value="findX3DModel">
        findX3DModel
        <br>
        <input type="radio" name="webmethod" value="getX3DModel">
        getX3DModel
        <br>
        <input type="radio" name="webmethod" value="findDESModel">
        findDESModel
        <br>
        <input type="radio" name="webmethod" value="getDESModel">
        getDESModel
        <br>
        <br>
        <input type="text" name="input">
        <input type="submit" value="Invoke Web Service">
        </form>
    </body>
</html>
```

b. *SavageWSClientServlet.jsp*

```
<%--
   Document   : SavageWSClientServlet
   Created on : Oct 13, 2008, 11:42:29 PM
   Author     : Leong, Hoe Wai
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">


<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
        charset=UTF-8">
        <title>Savage Web Services Test</title>
    </head>
<body>

    <%
    java.util.Properties configuration =
            new java.util.Properties();
    try {
          configuration.load(new java.io.FileInputStream
                  ("C:/Users/Lenovo/Documents/MDTS/NPS/Thesis/" +
                  "src/SavageWSClientWebAppDevTest/build/web/" +
                  "loaddir.properties"));
        } catch (Exception ex) {

        }
    if (request.getParameter("webmethod").compareTo
            ("findX3DModel")==0)
        try {
            x3dwsmethod.X3DWebServiceService service =
                    new x3dwsmethod.X3DWebServiceService();
            x3dwsmethod.X3DWebService port =
                    service.getX3DWebServicePort();
            java.lang.String searchTerm =
                    request.getParameter("input");
            java.lang.String result =
                    port.findX3DModel(searchTerm);
            // Create file
            java.io.FileWriter fstream =
                    new java.io.FileWriter(configuration.get
                    ("buildWebFilePath")+
                    "findX3DModelResult.xml");
            java.io.BufferedWriter output =
                    new java.io.BufferedWriter(fstream);
            output.write(result);
            //Close the output stream
            output.close();
            out.println
               ("<h1><A HREF=" +
               "\"findX3DModelResult.xml\" target=\"_blank\">" +
               "Savage Web Service Invocation Result</A></h1>");
        } catch (Exception ex) {
            System.out.println("EXCEPTION@findX3DModel : "+ex);
        }
    else if (request.getParameter("webmethod").
            compareTo("getX3DModel")==0)
        try {
            x3dwsmethod.X3DWebServiceService service =
                    new x3dwsmethod.X3DWebServiceService();
            x3dwsmethod.X3DWebService port =
                    service.getX3DWebServicePort();
            java.lang.String url =
                    request.getParameter("input");
            java.lang.String result = port.getX3DModel(url);
            // Create file
            java.io.FileWriter fstream =
                    new java.io.FileWriter(configuration.get
                    ("buildWebFilePath")+
                    "getX3DModelResult.x3d");
            java.io.BufferedWriter output =
                    new java.io.BufferedWriter(fstream);
            output.write(result);
            //Close the output stream
            output.close();
            out.println
              ("<h1><A HREF=\"getX3DModelResult.x3d\" target=" +
              "\"_blank\">Savage Web Service Invocation Result" +
              "</A></h1>");
        } catch (Exception ex) {
              System.out.println("EXCEPTION@getX3DModel : "+ex);
        }
    else if (request.getParameter("webmethod").compareTo
            ("findDESModel")==0)
        try {
            deswsmethod.DESWebServiceService service =
                    new deswsmethod.DESWebServiceService();
            deswsmethod.DESWebService port =
                    service.getDESWebServicePort();
```

```
                java.lang.String x3DUrl =
                        request.getParameter("input");
                java.lang.String result = port.findDESModel(x3DUrl);
                // Create file
                java.io.FileWriter fstream = new java.io.FileWriter
                        (configuration.get("buildWebFilePath")+
                        "findDESModelResult.xml");
                java.io.BufferedWriter output =
                        new java.io.BufferedWriter(fstream);
                output.write(result);
                //Close the output stream
                output.close();
                out.println
                 ("<h1><A HREF=\"findDESModelResult.xml\" target=" +
                 "\"_blank\">Savage Web Service Invocation Result" +
                 "</A></h1>");
            } catch (Exception ex) {
                  System.out.println("EXCEPTION@findDESModel : "+ex);
            }
        else if (request.getParameter("webmethod").
                            compareTo("getDESModel")==0)
            try {
                deswsmethod.DESWebServiceService service =
                        new deswsmethod.DESWebServiceService();
                deswsmethod.DESWebService port =
                        service.getDESWebServicePort();
                java.lang.String desUrl =
                        request.getParameter("input");
                java.lang.String result =
                        port.getDESModel(desUrl);
                // Create file
                java.io.FileWriter fstream =
                        new java.io.FileWriter(configuration.get
                        ("buildWebFilePath")+
                        "getDESModelResult.xml");
                java.io.BufferedWriter output =
                        new java.io.BufferedWriter(fstream);
                output.write(result);
                //Close the output stream
                output.close();
                out.println
                 ("<h1><A HREF=\"getDESModelResult.xml\" target=" +
                 "\"_blank\">Savage Web Service Invocation Result" +
                 "</A></h1>");
            } catch (Exception ex) {
                  System.out.println
                          ("EXCEPTION@getDESModel : "+ex);
            }
       %>
            <h2><A HREF=
            "http://localhost:9090/SavageWSClientWebAppDevTest/">
                Return to Query Page</A></h2>
    </body>
    </html>
```

APPENDIX B. MEDIATION FOR SAVAGE WEB SERVICES

1. OWL-S FOR SAVAGE COMPOSITE PROCESS

The SAVAGE composite process for OWL-S is illustrated in Chapter VI, section D. The services, service profiles, service groundings, atomic processes and composite process are created in Protégé OWL-S editor. The OWL-S description in XML is dynamically generated by Protégé. *SavageOWLSSemanticWS.owl* is the description for SAVAGE composite process. Each atomic process has its associated OWL description.

a. SavageOWLSSematicWS.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:service=
        "http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:process=
        "http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns="http://www.owl-ontologies.com/Ontology1220745514.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:list=
    "http://www.daml.org/services/owl-s/1.2/generic/
        ObjectList.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:wi1="http://www.example.org/owls/X3DDESDecision.owl"
    xmlns:expr="http://www.daml.org/services/owl-s/1.2/
                    generic/Expression.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:grounding=
    "http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:profile=
    "http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:time="http://www.isi.edu/~pan/damltime/time-entry.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1220745514.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource=
        "http://www.example.org/owls/X3DDESDecision.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Service.owl"/>
    <owl:imports rdf:resource=
        "http://www.example.org/owls/findX3DModel.owl"/>
    <owl:imports rdf:resource=
        "http://www.w3.org/2003/11/swrl"/>
    <owl:imports rdf:resource=
        "http://www.example.org/owls/getDESModel.owl"/>
    <owl:imports rdf:resource=
        "http://www.w3.org/2003/11/swrlb"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
    <owl:imports rdf:resource=
        "http://www.example.org/owls/findDESModel.owl"/>
    <owl:imports rdf:resource=
        "http://www.example.org/owls/getX3DModel.owl"/>
  </owl:Ontology>
  <process:Perform rdf:ID="findX3DModelProcess">
    <process:process rdf:resource="http://www.example.org/owls/
        findX3DModel.owl#findX3DModelProcess"/>
    <process:hasDataFrom>
      <process:InputBinding rdf:ID="InputBinding_4">
        <process:valueSource>
          <process:ValueOf rdf:ID="ValueOf_5">
            <process:fromProcess>
              <process:Perform rdf:ID="X3DDESDecisionProcess">
                <process:process rdf:resource
                   ="http://www.example.org/owls/
                    X3DDESDecision.owl#X3DDESDecisionProcess"/>
              </process:Perform>
            </process:fromProcess>
            <process:theVar rdf:resource=
                "http://www.example.org/owls/
                X3DDESDecision.owl#X3DDESDecisionResult"/>
          </process:ValueOf>
        </process:valueSource>
        <process:toParam rdf:resource=
            "http://www.example.org/owls/
                findX3DModel.owl#searchTerm"/>
      </process:InputBinding>
    </process:hasDataFrom>
  </process:Perform>
  <process:Perform rdf:ID="findDESModelProcess">
    <process:process rdf:resource=
        "http://www.example.org/owls/
            findDESModel.owl#findDESModelProcess"/>
    <process:hasDataFrom>
      <process:InputBinding rdf:ID="InputBinding_23">
        <process:valueSource>
          <process:ValueOf rdf:ID="ValueOf_24">
            <process:fromProcess rdf:resource=
                "#findX3DModelProcess"/>
            <process:theVar rdf:resource=
                "http://www.example.org/owls/
                    findX3DModel.owl#return"/>
          </process:ValueOf>
        </process:valueSource>
        <process:toParam rdf:resource=
            "http://www.example.org/owls/
                findDESModel.owl#x3dUrl"/>
      </process:InputBinding>
    </process:hasDataFrom>
  </process:Perform>
  <process:InputBinding rdf:ID="InputBinding_16">
    <process:toParam rdf:resource=
        "http://www.example.org/owls/getDESModel.owl#desUrl"/>
    <process:valueSource>
      <process:ValueOf rdf:ID="ValueOf_17">
        <process:fromProcess rdf:resource="#findDESModelProcess"/>
        <process:theVar rdf:resource=
            "http://www.example.org/owls/findDESModel.owl#return"/>
      </process:ValueOf>
    </process:valueSource>
  </process:InputBinding>
  <process:Perform rdf:ID="PerformGetX3D">
    <process:process rdf:resource=
        "http://www.example.org/owls/
            getX3DModel.owl#getX3DModelProcess"/>
    <process:hasDataFrom>
      <process:InputBinding rdf:ID="InputBinding_10">
        <process:valueSource>
          <process:ValueOf rdf:ID="ValueOf_11">
            <process:theVar rdf:resource=
                "http://www.example.org/owls/
                    findX3DModel.owl#return"/>
            <process:fromProcess>
              <process:Perform rdf:ID="PerformFindX3D">
                <process:process rdf:resource=
                    "http://www.example.org/owls/
                        findX3DModel.owl#findX3DModelProcess"/>
              </process:Perform>
            </process:fromProcess>
          </process:ValueOf>
        </process:valueSource>
        <process:toParam rdf:resource=
            "http://www.example.org/owls/getX3DModel.owl#url"/>
      </process:InputBinding>
    </process:hasDataFrom>
  </process:Perform>
  <process:ControlConstructList rdf:ID="ControlConstructList_5">
    <list:rest>
      <process:ControlConstructList rdf:ID=
            "ControlConstructList_16">
        <list:first>
          <process:If-Then-Else rdf:ID="If-Then-Else_15">
            <process:ifCondition>
              <expr:Condition rdf:ID="test"/>
            </process:ifCondition>
          </process:If-Then-Else>
        </list:first>
        <list:rest>
          <process:ControlConstructList rdf:ID=
            "ControlConstructList_8">
            <list:rest rdf:resource=
                "http://www.daml.org/services/owl-s/1.2/
                    generic/ObjectList.owl#nil"/>
            <list:first rdf:resource="#PerformGetX3D"/>
          </process:ControlConstructList>
        </list:rest>
      </process:ControlConstructList>
    </list:rest>
    <list:first rdf:resource="#PerformFindX3D"/>
```

```
</process:ControlConstructList>
<process:Sequence rdf:ID="Sequence_11">
  <process:components>
    <process:ControlConstructList rdf:ID=
                  "ControlConstructList_13">
      <list:first rdf:resource="#findDESModelProcess"/>
      <list:rest>
        <process:ControlConstructList rdf:ID=
                  "ControlConstructList_15">
          <list:first>
            <process:Perform rdf:ID="getDESModelProcess">
              <process:hasDataFrom rdf:resource=
                  "#InputBinding_16"/>
              <process:process rdf:resource=
                  "http://www.example.org/owls/
                      getDESModel.owl#getDESModelProcess"/>
            </process:Perform>
          </list:first>
          <list:rest rdf:resource=
              "http://www.daml.org/services/owl-s/1.2/generic/
                  ObjectList.owl#nil"/>
        </process:ControlConstructList>
      </list:rest>
    </process:ControlConstructList>
  </process:components>
</process:Sequence>
<process:Sequence rdf:ID="Sequence_18">
  <process:components>
    <process:ControlConstructList rdf:ID=
          "ControlConstructList_20">
      <list:rest rdf:resource=
          "http://www.daml.org/services/owl-s/1.2/generic/
              ObjectList.owl#nil"/>
      <list:first>
        <process:Perform rdf:ID="getX3DModelProcess">
          <process:process rdf:resource=
              "http://www.example.org/owls/
                  getX3DModel.owl#getX3DModelProcess"/>
          <process:hasDataFrom>
            <process:InputBinding rdf:ID="InputBinding_21">
              <process:valueSource>
                <process:ValueOf rdf:ID="ValueOf_22">
                  <process:fromProcess rdf:resource="
                      #findX3DModelProcess"/>
                  <process:theVar rdf:resource=
                      "http://www.example.org/owls/
                          findX3DModel.owl#return"/>
                </process:ValueOf>
              </process:valueSource>
              <process:toParam rdf:resource=
                  "http://www.example.org/owls/
                      getX3DModel.owl#url"/>
            </process:InputBinding>
          </process:hasDataFrom>
        </process:Perform>
      </list:first>
    </process:ControlConstructList>
  </process:components>
</process:Sequence>
<process:ControlConstructList rdf:ID="ControlConstructList_3">
  <list:first rdf:resource="#X3DDESDecisionProcess"/>
  <list:rest>
    <process:ControlConstructList rdf:ID=
                      "ControlConstructList_4">
      <list:rest>
        <process:ControlConstructList rdf:ID=
                      "ControlConstructList_10">
          <list:first>
            <process:If-Then-Else rdf:ID="If-Then-Else_6">
              <process:ifCondition>
                <expr:Condition rdf:ID="isFindMatchDES">
                  <profile:hasParameter rdf:resource=
                      "http://www.example.org/owls/
                      X3DDESDecision.owl#isGetMatchingDES"/>
                </expr:Condition>
              </process:ifCondition>
              <process:then rdf:resource="#Sequence_11"/>
              <process:else rdf:resource="#Sequence_18"/>
            </process:If-Then-Else>
          </list:first>
          <list:rest rdf:resource=
              "http://www.daml.org/services/owl-s/1.2/generic/
                  ObjectList.owl#nil"/>
        </process:ControlConstructList>
      </list:rest>
      <list:first rdf:resource="#findX3DModelProcess"/>
    </process:ControlConstructList>
  </list:rest>
</process:ControlConstructList>
<rdf:Description rdf:about=
      "http://www.example.org/owls/
      getX3DModel.owl#getX3DModelProfile">
  <profile:serviceCategory>
    <profile:ServiceCategory rdf:ID=
        "ServiceCategory_getX3DModelProfile">
      <profile:code rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#byte"
      >1</profile:code>
      <profile:taxonomy rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#string"
      >get</profile:taxonomy>
      <profile:value rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#string"
      >X3D</profile:value>
      <profile:categoryName rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#string"
      >Savage Webservices</profile:categoryName>
    </profile:ServiceCategory>
  </profile:serviceCategory>
</rdf:Description>
<profile:ServiceCategory rdf:ID="ServiceCategory_25"/>
<profile:ServiceParameter rdf:ID="ServiceParameter_26"/>
<process:Sequence rdf:ID="Sequence_2">
  <process:components rdf:resource="#ControlConstructList_3"/>
  </process:Sequence>
  <process:CompositeProcess rdf:ID="Savage_X3D_DES_Composite">
    <rdfs:comment rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
    >This composite process defines the workflow composition
        between findX3DModel, getX3DModel, findDESModel and
        getDESModel webmethods.</rdfs:comment>
    <process:composedOf rdf:resource="#Sequence_2"/>
  </process:CompositeProcess>
  <process:Result rdf:ID="Result_27"/>
  <process:InputBinding rdf:ID="InputBinding_9"/>
  <profile:ServiceCategory rdf:ID="ServiceCategory_1"/>
  <expr:LogicLanguage rdf:ID="LogicLanguage_9"/>
  <rdf:Description rdf:about="http://www.example.org/owls/
                         getX3DModel.owl#getX3DModelService">
    <rdfs:comment rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string"
    ></rdfs:comment>
  </rdf:Description>
  <process:Sequence rdf:ID="Sequence_3">
    <process:components rdf:resource="#ControlConstructList_5"/>
  </process:Sequence>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)
    http://protege.stanford.edu -->
```

b. X3DDESDecision.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0=
        "http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.1=
        "http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:j.2=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:j.3=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:p5="http://www.example.org/owls/X3DDESDecision.owl#"
  xml:base="http://www.example.org/owls/X3DDESDecision.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Service.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Process.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
  </owl:Ontology>
  <j.1:Output rdf:ID="X3DDESDecisionResult">
```

```
    <j.1:parameterType rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.w3.org/2001/XMLSchema#string</j.1:parameterType>
    <rdfs:label rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string"
    >X3DDESDecisionResult</rdfs:label>
  </j.1:Output>
  <j.0:Service rdf:ID="X3DDESDecisionService">
    <j.0:presents>
      <j.3:Profile rdf:ID="X3DDESDecisionProfile">
        <j.0:presentedBy rdf:resource="#X3DDESDecisionService"/>
        <j.3:hasInput>
          <j.1:Input rdf:ID="isGetMatchingDES">
            <rdfs:label rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#string"
            >isGetMatchingDES</rdfs:label>
            <j.1:parameterType rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://www.w3.org/2001/XMLSchema#boolean
                </j.1:parameterType>
          </j.1:Input>
        </j.3:hasInput>
        <j.3:hasInput>
          <j.1:Input rdf:ID="isGetX3D">
            <j.1:parameterType rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://www.w3.org/2001/XMLSchema#boolean
                </j.1:parameterType>
            <rdfs:label rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#string"
            >isGetX3D</rdfs:label>
          </j.1:Input>
        </j.3:hasInput>
        <j.3:hasInput>
          <j.1:Input rdf:ID="searchString">
            <rdfs:label rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#string"
            >searchString</rdfs:label>
            <j.1:parameterType rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://www.w3.org/2001/XMLSchema#string
                </j.1:parameterType>
          </j.1:Input>
        </j.3:hasInput>
        <j.3:textDescription rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#string"
        >Auto generated from
            http://localhost:8080/SavageX3DDESComposite.asmx?WSDL
                </j.3:textDescription>
        <j.3:serviceName rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string"
        >X3DDESDecision</j.3:serviceName>
        <j.3:hasOutput rdf:resource="#X3DDESDecisionResult"/>
      </j.3:Profile>
    </j.0:presents>
    <j.0:supports>
      <j.2:WsdlGrounding rdf:ID="X3DDESDecisionGrounding">
        <j.2:hasAtomicProcessGrounding>
          <j.2:WsdlAtomicProcessGrounding rdf:ID=
                "X3DDESDecisionAtomicProcessGrounding">
            <j.2:owlsProcess>
              <j.1:AtomicProcess rdf:ID="X3DDESDecisionProcess">
                <j.1:hasInput rdf:resource="#isGetMatchingDES"/>
                <j.1:hasInput rdf:resource="#isGetX3D"/>
                <j.1:hasInput rdf:resource="#searchString"/>
                <rdfs:label rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
                >X3DDESDecisionProcess</rdfs:label>
                <j.1:hasOutput rdf:resource=
                    "#X3DDESDecisionResult"/>
                <j.0:describes rdf:resource=
                    "#X3DDESDecisionService"/>
              </j.1:AtomicProcess>
            </j.2:owlsProcess>
            <j.2:wsdlInput>
              <j.2:WsdlInputMessageMap>
                <j.2:owlsParameter rdf:resource="#searchString"/>
                <j.2:wsdlMessagePart rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:8080/SavageX3DDESComposite.asmx?
                    WSDL#searchString</j.2:wsdlMessagePart>
              </j.2:WsdlInputMessageMap>
            </j.2:wsdlInput>
            <j.2:wsdlOperation>
              <j.2:WsdlOperationRef>
                <j.2:operation rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:8080/SavageX3DDESComposite.asmx?
                    WSDL#X3DDESDecision</j.2:operation>
                <j.2:portType rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:8080/SavageX3DDESComposite.asmx?
                    WSDL#Service1Soap</j.2:portType>
              </j.2:WsdlOperationRef>
            </j.2:wsdlOperation>
            <j.2:wsdlInput>
              <j.2:WsdlInputMessageMap>
                <j.2:wsdlMessagePart rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:8080/SavageX3DDESComposite.asmx?
                    WSDL#isGetX3D</j.2:wsdlMessagePart>
                <j.2:owlsParameter rdf:resource="#isGetX3D"/>
              </j.2:WsdlInputMessageMap>
            </j.2:wsdlInput>
            <j.2:wsdlOutputMessage rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://X3DDESComposite.org/#X3DDESDecisionSoapOut
                </j.2:wsdlOutputMessage>
            <j.2:wsdlDocument rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://localhost:8080/SavageX3DDESComposite.asmx?WSDL
                </j.2:wsdlDocument>
            <j.2:wsdlInput>
              <j.2:WsdlInputMessageMap>
```

```
                <j.2:wsdlMessagePart rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:8080/SavageX3DDESComposite.asmx?
                    WSDL#isGetMatchingDES</j.2:wsdlMessagePart>
                <j.2:owlsParameter rdf:resource=
                    "#isGetMatchingDES"/>
              </j.2:WsdlInputMessageMap>
            </j.2:wsdlInput>
            <j.2:wsdlInputMessage rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://X3DDESComposite.org/#X3DDESDecisionSoapIn
                </j.2:wsdlInputMessage>
            <j.2:wsdlOutput>
              <j.2:WsdlOutputMessageMap>
                <j.2:wsdlMessagePart rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:8080/SavageX3DDESComposite.asmx?
                    WSDL#X3DDESDecisionResult</j.2:wsdlMessagePart>
                <j.2:owlsParameter rdf:resource=
                    "#X3DDESDecisionResult"/>
              </j.2:WsdlOutputMessageMap>
            </j.2:wsdlOutput>
          </j.2:WsdlAtomicProcessGrounding>
        </j.2:hasAtomicProcessGrounding>
        <j.0:supportedBy rdf:resource="#X3DDESDecisionService"/>
      </j.2:WsdlGrounding>
    </j.0:supports>
    <j.0:describedBy rdf:resource="#X3DDESDecisionProcess"/>
  </j.0:Service>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)
            http://protege.stanford.edu -->
```

c. findX3DModel.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:p2="http://www.example.org/owls/findX3DModel.owl#"
    xmlns:j.0=
        "http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.1=
        "http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:j.2=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:j.3=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.org/owls/findX3DModel.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Service.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Process.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
  </owl:Ontology>
  <j.2:WsdlAtomicProcessGrounding rdf:ID=
        "findX3DModelAtomicProcessGrounding">
    <j.2:wsdlDocument rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://localhost:9090/SAVAGEWebServices/X3DWebService?
        wsdl</j.2:wsdlDocument>
    <j.2:owlsProcess>
      <j.1:AtomicProcess rdf:ID=
            "findX3DModelProcess">
        <j.1:hasInput>
          <j.1:Input rdf:ID="searchTerm">
            <rdfs:label rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#string"
            >searchTerm</rdfs:label>
            <j.1:parameterType rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://www.w3.org/2001/XMLSchema#string
                </j.1:parameterType>
          </j.1:Input>
        </j.1:hasInput>
        <j.1:hasOutput>
          <j.1:Output rdf:ID="return">
            <rdfs:label rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#string"
            >return</rdfs:label>
            <j.1:parameterType rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://www.w3.org/2001/XMLSchema#string
                </j.1:parameterType>
          </j.1:Output>
        </j.1:hasOutput>
        <rdfs:label rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string"
        >findX3DModelProcess</rdfs:label>
        <j.0:describes>
          <j.0:Service rdf:ID="findX3DModelService">
            <j.0:presents>
              <j.3:Profile rdf:ID="findX3DModelProfile">
                <j.0:presentedBy rdf:resource=
                    "#findX3DModelService"/>
                <j.3:serviceName rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
                >findX3DModel</j.3:serviceName>
                <j.3:textDescription rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
                >Auto generated from
                    http://localhost:9090/SAVAGEWebServices/
                        X3DWebService?wsdl</j.3:textDescription>
                <j.3:hasInput rdf:resource="#searchTerm"/>
                <j.3:hasOutput rdf:resource="#return"/>
              </j.3:Profile>
            </j.0:presents>
            <j.0:supports>
              <j.2:WsdlGrounding rdf:ID="findX3DModelGrounding">
                <j.0:supportedBy rdf:resource=
                    "#findX3DModelService"/>
                <j.2:hasAtomicProcessGrounding rdf:resource=
                    "#findX3DModelAtomicProcessGrounding"/>
              </j.2:WsdlGrounding>
            </j.0:supports>
            <j.0:describedBy rdf:resource=
                "#findX3DModelProcess"/>
          </j.0:Service>
        </j.0:describes>
      </j.1:AtomicProcess>
    </j.2:owlsProcess>
    <j.2:wsdlInputMessage rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://X3DWSMethod/#findX3DModel</j.2:wsdlInputMessage>
    <j.2:wsdlOperation>
      <j.2:WsdlOperationRef>
        <j.2:operation rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#anyURI"
        >http://localhost:9090/SAVAGEWebServices/X3DWebService?
            wsdl#findX3DModel</j.2:operation>
        <j.2:portType rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#anyURI"
        >http://localhost:9090/SAVAGEWebServices/X3DWebService?
                wsdl#X3DWebServicePort</j.2:portType>
      </j.2:WsdlOperationRef>
    </j.2:wsdlOperation>
    <j.2:wsdlOutputMessage rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://X3DWSMethod/#findX3DModelResponse
        </j.2:wsdlOutputMessage>
    <j.2:wsdlInput>
      <j.2:WsdlInputMessageMap>
        <j.2:owlsParameter rdf:resource="#searchTerm"/>
        <j.2:wsdlMessagePart rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#anyURI"
        >http://localhost:9090/SAVAGEWebServices/X3DWebService?
            wsdl#searchTerm</j.2:wsdlMessagePart>
      </j.2:WsdlInputMessageMap>
    </j.2:wsdlInput>
    <j.2:wsdlOutput>
      <j.2:WsdlOutputMessageMap>
        <j.2:wsdlMessagePart rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#anyURI"
        >http://localhost:9090/SAVAGEWebServices/X3DWebService?
            wsdl#return</j.2:wsdlMessagePart>
        <j.2:owlsParameter rdf:resource="#return"/>
      </j.2:WsdlOutputMessageMap>
    </j.2:wsdlOutput>
  </j.2:WsdlAtomicProcessGrounding>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)
            http://protege.stanford.edu -->
```

d. *getX3DModel.owl*

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0=
        "http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:p1="http://www.example.org/owls/getX3DModel.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.1=
        "http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:j.2=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:j.3=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.org/owls/getX3DModel.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Service.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Process.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
  </owl:Ontology>
  <j.0:Service rdf:ID="getX3DModelService">
    <j.0:supports>
      <j.2:WsdlGrounding rdf:ID="getX3DModelGrounding">
        <j.0:supportedBy rdf:resource="#getX3DModelService"/>
        <j.2:hasAtomicProcessGrounding>
          <j.2:WsdlAtomicProcessGrounding rdf:ID=
                "getX3DModelAtomicProcessGrounding">
            <j.2:wsdlOutputMessage rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://X3DWSMethod/#getX3DModelResponse
                </j.2:wsdlOutputMessage>
            <j.2:wsdlInput>
              <j.2:WsdlInputMessageMap>
                <j.2:wsdlMessagePart rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:9090/SAVAGEWebServices/
                    X3DWebService?wsdl#url</j.2:wsdlMessagePart>
                <j.2:owlsParameter>
                  <j.1:Input rdf:ID="url">
                    <j.1:parameterType rdf:datatype=
                        "http://www.w3.org/2001/XMLSchema#anyURI"
                    >http://www.w3.org/2001/XMLSchema#string
                        </j.1:parameterType>
                    <rdfs:label rdf:datatype=
                        "http://www.w3.org/2001/XMLSchema#string"
                    >url</rdfs:label>
                  </j.1:Input>
                </j.2:owlsParameter>
              </j.2:WsdlInputMessageMap>
            </j.2:wsdlInput>
            <j.2:wsdlOutput>
              <j.2:WsdlOutputMessageMap>
                <j.2:wsdlMessagePart rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:9090/SAVAGEWebServices/
                    X3DWebService?wsdl#return</j.2:wsdlMessagePart>
                <j.2:owlsParameter>
                  <j.1:Output rdf:ID="return">
                    <rdfs:label rdf:datatype=
                        "http://www.w3.org/2001/XMLSchema#string"
                    >return</rdfs:label>
                    <j.1:parameterType rdf:datatype=
                        "http://www.w3.org/2001/XMLSchema#anyURI"
                    >http://www.w3.org/2001/XMLSchema#string
                        </j.1:parameterType>
                  </j.1:Output>
                </j.2:owlsParameter>
              </j.2:WsdlOutputMessageMap>
            </j.2:wsdlOutput>
            <j.2:wsdlDocument rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://localhost:9090/SAVAGEWebServices/X3DWebService
                ?wsdl</j.2:wsdlDocument>
            <j.2:owlsProcess>
              <j.1:AtomicProcess rdf:ID="getX3DModelProcess">
                <j.1:hasInput rdf:resource="#url"/>
                <j.0:describes rdf:resource="#getX3DModelService"/>
                <rdfs:label rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
                >getX3DModelProcess</rdfs:label>
                <j.1:hasOutput rdf:resource="#return"/>
              </j.1:AtomicProcess>
            </j.2:owlsProcess>
            <j.2:wsdlOperation>
              <j.2:WsdlOperationRef>
                <j.2:portType rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:9090/SAVAGEWebServices/
                    X3DWebService?wsdl#X3DWebServicePort
                        </j.2:portType>
                <j.2:operation rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#anyURI"
                >http://localhost:9090/SAVAGEWebServices/
                    X3DWebService?wsdl#getX3DModel</j.2:operation>
              </j.2:WsdlOperationRef>
            </j.2:wsdlOperation>
            <j.2:wsdlInputMessage rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#anyURI"
            >http://X3DWSMethod/#getX3DModel</j.2:wsdlInputMessage>
          </j.2:WsdlAtomicProcessGrounding>
        </j.2:hasAtomicProcessGrounding>
      </j.2:WsdlGrounding>
    </j.0:supports>
    <j.0:describedBy rdf:resource="#getX3DModelProcess"/>
    <j.0:presents>
```

```
      <j.3:Profile rdf:ID="getX3DModelProfile">
        <j.0:presentedBy rdf:resource="#getX3DModelService"/>
        <j.3:textDescription rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string"
        >Auto generated from
            http://localhost:9090/SAVAGEWebServices/
            X3DWebService?wsdl</j.3:textDescription>
        <j.3:hasOutput rdf:resource="#return"/>
        <j.3:hasInput rdf:resource="#url"/>
        <j.3:serviceName rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string"
        >getX3DModel</j.3:serviceName>
      </j.3:Profile>
    </j.0:presents>
  </j.0:Service>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)
        http://protege.stanford.edu -->
```

e. findDESModel.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:p3="http://www.example.org/owls/findDESModel.owl#"
    xmlns:j.0=
        "http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.1=
        "http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:j.2=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:j.3=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.org/owls/findDESModel.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Service.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Process.owl"/>
    <owl:imports rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
  </owl:Ontology>
  <j.1:Output rdf:ID="return">
    <j.1:parameterType rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.w3.org/2001/XMLSchema#string
        </j.1:parameterType>
    <rdfs:label rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string"
    >return</rdfs:label>
```

```
</j.1:Output>
<j.2:WsdlAtomicProcessGrounding rdf:ID=
      "findDESModelAtomicProcessGrounding">
  <j.2:wsdlOperation>
    <j.2:WsdlOperationRef>
      <j.2:operation rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
      >http://localhost:9090/SAVAGEWebServices/DESWebService
          ?wsdl#findDESModel</j.2:operation>
      <j.2:portType rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
      >http://localhost:9090/SAVAGEWebServices/DESWebService?
          wsdl#DESWebServicePort</j.2:portType>
    </j.2:WsdlOperationRef>
  </j.2:wsdlOperation>
  <j.2:wsdlInputMessage rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
  >http://DESWSMethod/#findDESModel</j.2:wsdlInputMessage>
  <j.2:wsdlInput>
    <j.2:WsdlInputMessageMap>
      <j.2:owlsParameter>
        <j.1:Input rdf:ID="x3dUrl">
          <j.1:parameterType rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#anyURI"
          >http://www.w3.org/2001/XMLSchema#string
              </j.1:parameterType>
          <rdfs:label rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#string"
          >x3dUrl</rdfs:label>
        </j.1:Input>
      </j.2:owlsParameter>
      <j.2:wsdlMessagePart rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
      >http://localhost:9090/SAVAGEWebServices/DESWebService
          ?wsdl#x3dUrl</j.2:wsdlMessagePart>
    </j.2:WsdlInputMessageMap>
  </j.2:wsdlInput>
  <j.2:wsdlOutputMessage rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
  >http://DESWSMethod/#findDESModelResponse
          </j.2:wsdlOutputMessage>
  <j.2:wsdlOutput>
    <j.2:WsdlOutputMessageMap>
      <j.2:wsdlMessagePart rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
      >http://localhost:9090/SAVAGEWebServices/DESWebService
          ?wsdl#return</j.2:wsdlMessagePart>
      <j.2:owlsParameter rdf:resource="#return"/>
    </j.2:WsdlOutputMessageMap>
  </j.2:wsdlOutput>
  <j.2:wsdlDocument rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#anyURI"
  >http://localhost:9090/SAVAGEWebServices/DESWebService?
      wsdl</j.2:wsdlDocument>
  <j.2:owlsProcess>
    <j.1:AtomicProcess rdf:ID="findDESModelProcess">
      <rdfs:label rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#string"
        >findDESModelProcess</rdfs:label>
        <j.0:describes>
          <j.0:Service rdf:ID="findDESModelService">
            <j.0:supports>
              <j.2:WsdlGrounding rdf:ID="findDESModelGrounding">
                <j.2:hasAtomicProcessGrounding rdf:resource=
                    "#findDESModelAtomicProcessGrounding"/>
                <j.0:supportedBy rdf:resource=
                    "#findDESModelService"/>
              </j.2:WsdlGrounding>
            </j.0:supports>
            <j.0:presents>
              <j.3:Profile rdf:ID="findDESModelProfile">
                <j.3:textDescription rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
                >Auto generated from
                    http://localhost:9090/SAVAGEWebServices/
                    DESWebService?wsdl</j.3:textDescription>
                <j.3:hasInput rdf:resource="#x3dUrl"/>
                <j.3:hasOutput rdf:resource="#return"/>
                <j.3:serviceName rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#string"
                >findDESModel</j.3:serviceName>
                <j.0:presentedBy rdf:resource=
                    "#findDESModelService"/>
              </j.3:Profile>
            </j.0:presents>
            <j.0:describedBy rdf:resource=
                "#findDESModelProcess"/>
          </j.0:Service>
        </j.0:describes>
        <j.1:hasInput rdf:resource="#x3dUrl"/>
        <j.1:hasOutput rdf:resource="#return"/>
      </j.1:AtomicProcess>
    </j.2:owlsProcess>
  </j.2:WsdlAtomicProcessGrounding>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)
        http://protege.stanford.edu -->
```

f. *getDESModel.owl*

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0=
        "http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.1=
        "http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:p4="http://www.example.org/owls/getDESModel.owl#"
    xmlns:j.2=
        "http://www.daml.org/services/owl-s/1.2/Grounding.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
```

```
  xmlns:j.3=
      "http://www.daml.org/services/owl-s/1.2/Profile.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.example.org/owls/getDESModel.owl">
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource=
      "http://www.daml.org/services/owl-s/1.2/Service.owl"/>
  <owl:imports rdf:resource=
      "http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
  <owl:imports rdf:resource=
      "http://www.daml.org/services/owl-s/1.2/Process.owl"/>
  <owl:imports rdf:resource=
      "http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
</owl:Ontology>
<j.0:Service rdf:ID="getDESModelService">
  <j.0:presents>
    <j.3:Profile rdf:ID="getDESModelProfile">
      <j.3:hasInput>
        <j.1:Input rdf:ID="desUrl">
          <j.1:parameterType rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#anyURI"
          >http://www.w3.org/2001/XMLSchema#string
              </j.1:parameterType>
          <rdfs:label rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#string"
          >desUrl</rdfs:label>
        </j.1:Input>
      </j.3:hasInput>
      <j.3:hasOutput>
        <j.1:Output rdf:ID="return">
          <rdfs:label rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#string"
          >return</rdfs:label>
          <j.1:parameterType rdf:datatype=
              "http://www.w3.org/2001/XMLSchema#anyURI"
          >http://www.w3.org/2001/XMLSchema#string
              </j.1:parameterType>
        </j.1:Output>
      </j.3:hasOutput>
      <j.0:presentedBy rdf:resource="#getDESModelService"/>
      <j.3:serviceName rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#string"
      >getDESModel</j.3:serviceName>
      <j.3:textDescription rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#string"
      >Auto generated from
          http://localhost:9090/SAVAGEWebServices/
              DESWebService?wsdl</j.3:textDescription>
    </j.3:Profile>
  </j.0:presents>
  <j.0:supports>
    <j.2:WsdlGrounding rdf:ID="getDESModelGrounding">
      <j.2:hasAtomicProcessGrounding>
        <j.2:WsdlAtomicProcessGrounding rdf:ID=
            "getDESModelAtomicProcessGrounding">
          <j.2:wsdlOutput>
            <j.2:WsdlOutputMessageMap>
    <j.2:wsdlMessagePart rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://localhost:9090/SAVAGEWebServices/
        DESWebService?wsdl#return</j.2:
            wsdlMessagePart>
    <j.2:owlsParameter rdf:resource="#return"/>
   </j.2:WsdlOutputMessageMap>
  </j.2:wsdlOutput>
  <j.2:wsdlDocument rdf:datatype=
       "http://www.w3.org/2001/XMLSchema#anyURI"
  >http://localhost:9090/SAVAGEWebServices/DESWebService
        ?wsdl</j.2:wsdlDocument>
  <j.2:wsdlInput>
   <j.2:WsdlInputMessageMap>
    <j.2:owlsParameter rdf:resource="#desUrl"/>
    <j.2:wsdlMessagePart rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://localhost:9090/SAVAGEWebServices/
        DESWebService?wsdl#desUrl</j.2:
            wsdlMessagePart>
   </j.2:WsdlInputMessageMap>
  </j.2:wsdlInput>
  <j.2:wsdlOutputMessage rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#anyURI"
  >http://DESWSMethod/#getDESModelResponse
      </j.2:wsdlOutputMessage>
  <j.2:owlsProcess>
   <j.1:AtomicProcess rdf:ID="getDESModelProcess">
    <j.1:hasInput rdf:resource="#desUrl"/>
    <rdfs:label rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string"
    >getDESModelProcess</rdfs:label>
    <j.1:hasOutput rdf:resource="#return"/>
    <j.0:describes rdf:resource=
        "#getDESModelService"/>
   </j.1:AtomicProcess>
  </j.2:owlsProcess>
  <j.2:wsdlInputMessage rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#anyURI"
  >http://DESWSMethod/#getDESModel
      </j.2:wsdlInputMessage>
  <j.2:wsdlOperation>
   <j.2:WsdlOperationRef>
    <j.2:operation rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://localhost:9090/SAVAGEWebServices/
        DESWebService?wsdl#getDESModel</j.2:operation>
    <j.2:portType rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#anyURI"
    >http://localhost:9090/SAVAGEWebServices/
        DESWebService?wsdl#DESWebServicePort
            </j.2:portType>
   </j.2:WsdlOperationRef>
  </j.2:wsdlOperation>
 </j.2:WsdlAtomicProcessGrounding>
</j.2:hasAtomicProcessGrounding>
<j.0:supportedBy rdf:resource="#getDESModelService"/>
```

```
      </j.2:WsdlGrounding>
    </j.0:supports>
    <j.0:describedBy rdf:resource="#getDESModelProcess"/>
  </j.0:Service>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.2.1, Build 365)
             http://protege.stanford.edu -->
```

2. WSBPEL FOR SAVAGE COMPOSITE PROCESS

The WSBPEL composite process for SAVAGE web services is constructed using NetBeans. NetBeans dynamically generates WSBPEL description in XML. compositeProcess.bpel describes the composite process. The remaning WSDL files are components required for the composite process. WSDL files for X3D and DES web services, which are listed in Appendix A, are also used in WSBPEL composite process.

a. compositeProcess.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="compositeProcess"
    targetNamespace=
        "http://enterprise.netbeans.org/bpel/
                BpelModule1/compositeProcess"
    xmlns=
    "http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns=
    "http://enterprise.netbeans.org/bpel/BpelModule1/
        compositeProcess"
        xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/
                    executable/SUNExtension/Trace"
        xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/
                    executable/SUNExtension/Editor"
        xmlns:ns0="http://docs.oasis-open.org/wsbpel/2.0/
                    process/executable">
    <import namespace=
"http://enterprise.netbeans.org/bpel/X3DWebServiceServiceWrapper"
        location="X3DWebServiceServiceWrapper.wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://X3DWSMethod/"
        location="X3DWebServiceService.wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace=
    "http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
    location="SavageBpelX3DDESSeqComposite.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace=
    "http://j2ee.netbeans.org/wsdl/SavageGetX3DDESDecision"
    location="SavageGetX3DDESDecision.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace=
"http://enterprise.netbeans.org/bpel/X3DWebServiceServiceWrapper"
    location="X3DWebServiceServiceWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace=
"http://enterprise.netbeans.org/bpel/DESWebServiceServiceWrapper"
    location="DESWebServiceServiceWrapper.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://DESWSMethod/"
    location="DESWebServiceService.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink name="SavageBpelX3DDESSeqLink"
        xmlns:tns=
"http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
        partnerLinkType="tns:SavageBpelX3DDESSeqComposite"
        myRole="SavageBpelX3DDESSeqCompositePortTypeRole"/>
        <partnerLink name="X3DWebServiceLinkForComposite"
        xmlns:tns=
  "http://enterprise.netbeans.org/bpel/X3DWebServiceServiceWrapper"
        partnerLinkType="tns:X3DWebServiceLinkForComposite"
        myRole="X3DWebServiceRole"
        partnerRole="X3DWebPartnerServiceRole"/>
        <partnerLink name="DESWSLinkComposite"
        xmlns:tns=
   "http://enterprise.netbeans.org/bpel/DESWebServiceServiceWrapper"
        partnerLinkType="tns:DESWSLinkComposite"
        myRole="DESWebServiceRole"
        partnerRole="DESWebPartnerServiceRole"/>
        <partnerLink name="SavageDESX3DDecision"
        xmlns:tns=
            "http://j2ee.netbeans.org/wsdl/SavageGetX3DDESDecision"
            partnerLinkType="tns:SavageGetX3DDESDecision"
            myRole="SavageGetX3DDESDecisionPortTypeRole"/>
    </partnerLinks>
    <variables>
        <variable
            name="GetDESInvokeOut"
            xmlns:tns="http://DESWSMethod/"
            messageType="tns:getDESModelResponse"/>
        <variable
            name="GetDESInvokeIn"
            xmlns:tns="http://DESWSMethod/"
            messageType="tns:getDESModel"/>
        <variable
            name="GetX3DInvokeOut"
            xmlns:tns="http://X3DWSMethod/"
            messageType="tns:getX3DModelResponse"/>
        <variable
            name="GetX3DInvokeIn"
            xmlns:tns="http://X3DWSMethod/"
            messageType="tns:getX3DModel"/>
        <variable
            name="FindDESInvokeOut"
            xmlns:tns="http://DESWSMethod/"
            messageType="tns:findDESModelResponse"/>
        <variable
            name="FindDESInvokeIn"
            xmlns:tns="http://DESWSMethod/"
            messageType="tns:findDESModel"/>
        <variable
            name="FindX3DInvokeOut"
            xmlns:tns="http://X3DWSMethod/"
            messageType="tns:findX3DModelResponse"/>
        <variable name="FindX3DInvokeIn"
            xmlns:tns="http://X3DWSMethod/"
            messageType="tns:findX3DModel"/>
        <variable name="SavageBpelX3DDESSeqCompositeOperationOut"
        xmlns:tns=
    "http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
        messageType=
            "tns:SavageBpelX3DDESSeqCompositeOperationResponse"/>
        <variable name="SavageBpelX3DDESSeqCompositeOperationIn"
       xmlns:tns=
       "http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
       messageType=
        "tns:SavageBpelX3DDESSeqCompositeOperationRequest"/>
    </variables>
    <sequence>
        <receive name="startComposite" createInstance="yes"
        partnerLink="SavageBpelX3DDESSeqLink"
        operation="SavageBpelX3DDESSeqCompositeOperation"
        xmlns:tns=

"http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
        portType="tns:SavageBpelX3DDESSeqCompositePortType"
        variable="SavageBpelX3DDESSeqCompositeOperationIn"/>
        <assign name="Assign1">
            <copy>
                <from variable=
                "SavageBpelX3DDESSeqCompositeOperationIn"
                part="searchString"/>
                <to>$FindX3DInvokeIn.parameters/searchTerm</to>
            </copy>
        </assign>
        <invoke name="InvokeX3DFind"
            partnerLink="X3DWebServiceLinkForComposite"
            operation="findX3DModel"
            xmlns:tns="http://X3DWSMethod/"
            portType="tns:X3DWebService"
            inputVariable="FindX3DInvokeIn"
            outputVariable="FindX3DInvokeOut"/>
        <assign name="Assign2">
            <copy>
                <from>ns0:doXslTransform
                    ('urn:stylesheets:transformX3DUrlList',
                $FindX3DInvokeOut.parameters/return)</from>
                <to>$GetX3DInvokeIn.parameters/url</to>
            </copy>
            <copy>
                <from>ns0:doXslTransform('urn:stylesheets:
                transformX3DUrlList',$FindX3DInvokeOut.
                parameters/return)</from>
                <to variable="FindDESInvokeIn" part="parameters"/>
            </copy>
        </assign>
        <if name="If">
            <condition>
                $SavageBpelX3DDESSeqCompositeOperationIn.
                    isGetX3DModel</condition>
            <invoke name="InvokeX3DGet"
            partnerLink="X3DWebServiceLinkForComposite"
            operation="getX3DModel"
            xmlns:tns="http://X3DWSMethod/"
            portType="tns:X3DWebService"
            inputVariable="GetX3DInvokeIn"
            outputVariable="GetX3DInvokeOut"/>
            <elseif>
                <condition>
                    $SavageBpelX3DDESSeqCompositeOperationIn.
                    isFindGetDESModel</condition>
                <sequence name="Sequence1">
                    <invoke name="InvokeFindDES"
                    partnerLink="DESWSLinkComposite"
                    operation="findDESModel"
                    xmlns:tns="http://DESWSMethod/"
                    portType="tns:DESWebService"
                    inputVariable="FindDESInvokeIn"
                    outputVariable="FindDESInvokeOut"/>
                    <assign name="Assign3">
                        <copy>
                            <from>$FindDESInvokeOut.
                            parameters/return</from>
                            <to>$GetDESInvokeIn.
                            parameters/desUrl</to>
                        </copy>
                    </assign>
                    <invoke name="InvokeGetDES"
                    partnerLink="DESWSLinkComposite"
                    operation="getDESModel"
                    xmlns:tns="http://DESWSMethod/"
                    portType="tns:DESWebService"
                    inputVariable="GetDESInvokeIn"
                    outputVariable="GetDESInvokeOut"/>
                </sequence>
            </elseif>
            <else>
                <empty name="doNothing"/>
            </else>
        </if>
        <reply name="endComposite"
            partnerLink="SavageBpelX3DDESSeqLink"
            operation="SavageBpelX3DDESSeqCompositeOperation"
            xmlns:tns=
       "http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
```

```
        portType="tns:SavageBpelX3DDESSeqCompositePortType"
        variable="SavageBpelX3DDESSeqCompositeOperationOut"/>
    </sequence>
</process>
```

b. DESWebServiceServiceWrapper.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        name="DESWebServiceServiceWrapper"
        targetNamespace=
        "http://enterprise.netbeans.org/bpel/
            DESWebServiceServiceWrapper"
        xmlns:tns=
        "http://enterprise.netbeans.org/bpel/
        DESWebServiceServiceWrapper"
        xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
        xmlns:ns="http://DESWSMethod/">
    <import location="DESWebServiceService.wsdl"
    namespace="http://DESWSMethod/"/>
    <plnk:partnerLinkType
        name="DESWSLinkComposite">
        <plnk:role name="DESWebServiceRole"
            portType="ns:DESWebService"/>
        <plnk:role name="DESWebPartnerServiceRole"
            portType="ns:DESWebService"/>
    </plnk:partnerLinkType>
</definitions>
```

c. SavageBpelX3DDESSeqComposite.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="SavageBpelX3DDESSeqComposite"
    targetNamespace=
    "http://j2ee.netbeans.org/wsdl/SavageBpelX3DDESSeqComposite"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://j2ee.netbeans.org/wsdl/
    SavageBpelX3DDESSeqComposite"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types/>
    <message name="SavageBpelX3DDESSeqCompositeOperationRequest">
        <part name="searchString" type="xsd:string"/>
        <part name="isGetX3DModel" type="xsd:boolean"/>
        <part name="isFindGetDESModel" type="xsd:boolean"/>
    </message>
    <message name="SavageBpelX3DDESSeqCompositeOperationResponse">
        <part name="resultString" type="xsd:string"/>
    </message>
    <portType name="SavageBpelX3DDESSeqCompositePortType">
        <operation name="SavageBpelX3DDESSeqCompositeOperation">
            <input name="input1"
                message=
                "tns:SavageBpelX3DDESSeqCompositeOperationRequest"/>
            <output name="output1"
                message=

"tns:SavageBpelX3DDESSeqCompositeOperationResponse"/>
        </operation>
    </portType>
    <binding name="SavageBpelX3DDESSeqCompositeBinding"
            type="tns:SavageBpelX3DDESSeqCompositePortType">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="SavageBpelX3DDESSeqCompositeOperation">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"
                namespace=
                "http://j2ee.netbeans.org/wsdl/
                SavageBpelX3DDESSeqComposite"/>
            </input>
            <output name="output1">
                <soap:body use="literal"
                namespace="http://j2ee.netbeans.org/wsdl/
                SavageBpelX3DDESSeqComposite"/>
            </output>
        </operation>
    </binding>
    <service name="SavageBpelX3DDESSeqCompositeService">
        <port name="SavageBpelX3DDESSeqCompositePort"
            binding="tns:SavageBpelX3DDESSeqCompositeBinding">
            <soap:address location=
            "http://localhost:${HttpDefaultPort}/
            SavageBpelX3DDESSeqCompositeService/
            SavageBpelX3DDESSeqCompositePort"/>
        </port>
    </service>
    <plnk:partnerLinkType name="SavageBpelX3DDESSeqComposite">
        <!-- A partner link type is automatically generated
        when a new port type is added. Partner link types are
        used by BPEL processes.
        In a BPEL process, a partner link represents the
        interaction between the BPEL process and a partner
        service. Each partner link is associated with a
        partner link type.
        A partner link type characterizes the conversational
        relationship between two services. The partner link
        type can have one or two roles.-->
        <plnk:role
            name="SavageBpelX3DDESSeqCompositePortTypeRole"
            portType="tns:SavageBpelX3DDESSeqCompositePortType"/>
    </plnk:partnerLinkType>
</definitions>
```

d. SavageBpelX3DSeq.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="SavageBpelX3DSeq" targetNamespace
    ="http://j2ee.netbeans.org/wsdl/SavageBpelX3DSeq"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://j2ee.netbeans.org/wsdl/SavageBpelX3DSeq"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types/>
    <message name="SavageBpelX3DSeqOperationRequest">
        <part name="part1" type="xsd:string"/>
    </message>
    <message name="SavageBpelX3DSeqOperationResponse">
        <part name="part1" type="xsd:string"/>
    </message>
    <portType name="SavageBpelX3DSeqPortType">
        <operation name="SavageBpelX3DSeqOperation">
            <input name="input1"
                message="tns:SavageBpelX3DSeqOperationRequest"/>
            <output name="output1"
                message="tns:SavageBpelX3DSeqOperationResponse"/>
        </operation>
    </portType>
    <binding name="SavageBpelX3DSeqBinding"
        type="tns:SavageBpelX3DSeqPortType">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="SavageBpelX3DSeqOperation">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"
                    namespace=
               "http://j2ee.netbeans.org/wsdl/SavageBpelX3DSeq"/>
            </input>
            <output name="output1">
                <soap:body use="literal"
                namespace
                ="http://j2ee.netbeans.org/wsdl/SavageBpelX3DSeq"/>
            </output>
        </operation>
    </binding>
    <service name="SavageBpelX3DSeqService">
        <port name="SavageBpelX3DSeqPort"
            binding="tns:SavageBpelX3DSeqBinding">
            <soap:address location=
            "http://localhost:${HttpDefaultPort}
            /SavageBpelX3DSeqService/SavageBpelX3DSeqPort"/>
        </port>
    </service>
    <plnk:partnerLinkType name="SavageBpelX3DSeq">
        <!-- A partner link type is automatically generated
        when a new port type is added. Partner link types
        are used by BPEL processes. In a BPEL process, a partner
        link represents the interaction between the BPEL process
        and a partner service. Each partner link is associated
        with a partner link type.
        A partner link type characterizes the conversational
        relationship between two services. The partner link
        type can have one or two roles.-->
        <plnk:role name="SavageBpelX3DSeqPortTypeRole"
        portType="tns:SavageBpelX3DSeqPortType"/>
    </plnk:partnerLinkType>
</definitions>
```

e. SavageGetX3DDESDecision.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="SavageGetX3DDESDecision"
targetNamespace=
"http://j2ee.netbeans.org/wsdl/SavageGetX3DDESDecision"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns=
    "http://j2ee.netbeans.org/wsdl/SavageGetX3DDESDecision"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types/>
    <message name="SavageGetX3DDESDecisionOperationRequest">
        <part name="isMatchDES" type="xsd:boolean"/>
    </message>
    <message name="SavageGetX3DDESDecisionOperationResponse"/>
    <portType name="SavageGetX3DDESDecisionPortType">
        <operation name="SavageGetX3DDESDecisionOperation">
            <input name="input1"
            message=
            "tns:SavageGetX3DDESDecisionOperationRequest"/>
            <output name="output1"
            message=
            "tns:SavageGetX3DDESDecisionOperationResponse"/>
        </operation>
    </portType>
    <binding name="SavageGetX3DDESDecisionBinding"
    type="tns:SavageGetX3DDESDecisionPortType">
        <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="SavageGetX3DDESDecisionOperation">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"
                namespace=
         "http://j2ee.netbeans.org/wsdl/SavageGetX3DDESDecision"/>
            </input>
            <output name="output1">
                <soap:body use="literal"
                namespace=
          "http://j2ee.netbeans.org/wsdl/SavageGetX3DDESDecision"/>
            </output>
        </operation>
    </binding>
    <service name="SavageGetX3DDESDecisionService">
        <port name="SavageGetX3DDESDecisionPort"
        binding="tns:SavageGetX3DDESDecisionBinding">
            <soap:address
            location="http://localhost:${HttpDefaultPort}
            /SavageGetX3DDESDecisionService/
            SavageGetX3DDESDecisionPort"/>
        </port>
    </service>
    <plnk:partnerLinkType name="SavageGetX3DDESDecision">
        <!-- A partner link type is automatically generated
        when a new port type is added. Partner link types are
        used by BPEL processes. In a BPEL process, a partner link
        represents the interaction between the BPEL process and
        a partner service. Each partner link is associated with
        a partner link type. A partner link type characterizes
        the conversational relationship between two services.
        The partner link type can have one or two roles.-->
        <plnk:role name="SavageGetX3DDESDecisionPortTypeRole"
        portType="tns:SavageGetX3DDESDecisionPortType"/>
    </plnk:partnerLinkType>
</definitions>
```

f. X3DWebServiceServiceWrapper.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="X3DWebServiceServiceWrapper"
    targetNamespace=
    "http://enterprise.netbeans.org/bpel/
    X3DWebServiceServiceWrapper"
    xmlns:tns="http://enterprise.netbeans.org/bpel/
    X3DWebServiceServiceWrapper"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:ns="http://X3DWSMethod/">
    <import location="X3DWebServiceService.wsdl"
    namespace="http://X3DWSMethod/"/>
    <plnk:partnerLinkType name="X3DWebServiceLinkType">
        <plnk:role name="X3DWebServiceRole"
        portType="ns:X3DWebService"/>
        <plnk:role name="X3DWebPartnerServiceRole"
        portType="ns:X3DWebService"/>
    </plnk:partnerLinkType>
    <plnk:partnerLinkType name="X3DWebServiceLinkForComposite">
        <plnk:role name="X3DWebServiceRole"
        portType="ns:X3DWebService"/>
        <plnk:role name="X3DWebPartnerServiceRole"
        portType="ns:X3DWebService"/>
    </plnk:partnerLinkType>
</definitions>
```

g. transformX3DUrlList.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="text"/>
    <xsl:template match="/x3-dFind-result-entity">
        <xsl:for-each select="./result-set">
            <xsl:if test=".=../result-set[1]">
                <xsl:value-of select="./value"/>
                <xsl:text></xsl:text>
            </xsl:if>
        </xsl:for-each>
     </xsl:template>
</xsl:stylesheet>
```

APPENDIX C. RETRIEVE EXAMPLES

1. SAVAGE WEB SERVICES SOURCE CODE

Online at

https://savage.nps.edu/svn/nps/Savage/services/WebServices/SAVAGEWebServices

Subversion master source at

https://savage.nps.edu/svn/nps/Savage/services/WebServices/SAVAGEWebServices/src

War file at (~11.2 MB)

https://savage.nps.edu/svn/nps/Savage/services/WebServices/SAVAGEWebServices/dist/SAVAGEWebServices.war

2. SAVAGE WEB SERVICES UML DIAGRAMS

Online at

https://savage.nps.edu/svn/nps/Savage/services/WebServices/SAVAGEModel/report/index.html

Subversion master source at

https://savage.nps.edu/svn/nps/Savage/services/WebServices/SAVAGEModel/

3. JSP CLIENT DEMO URL

Online at

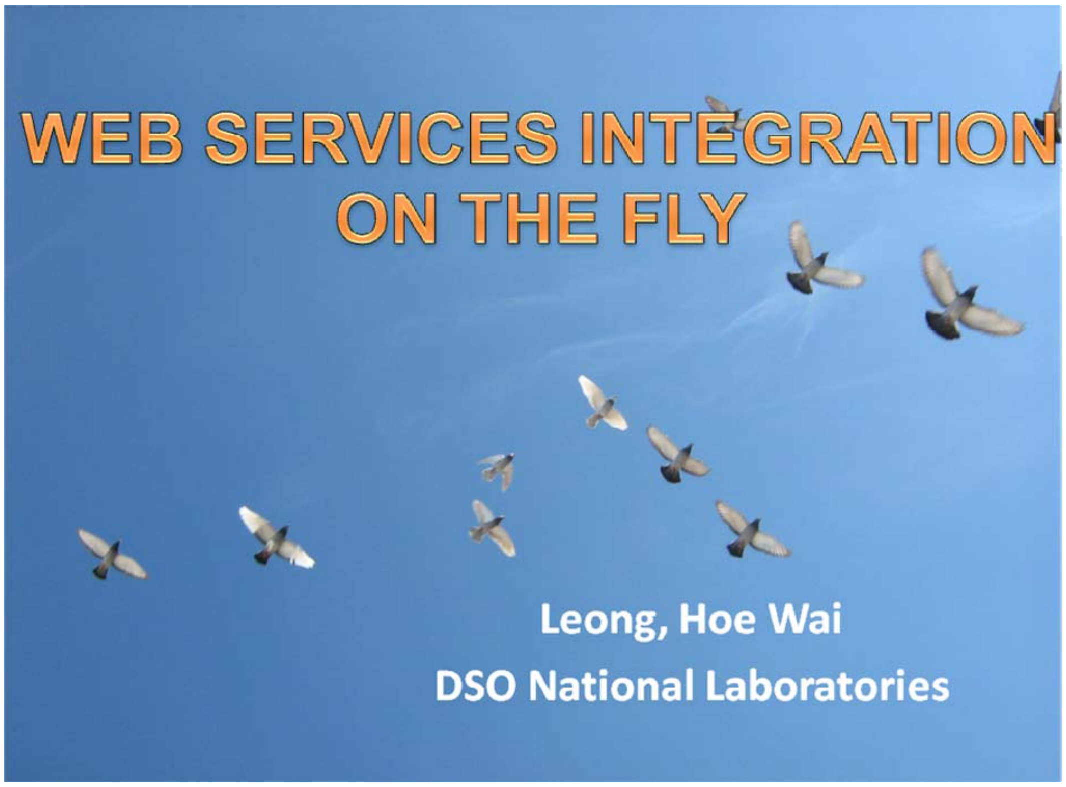
https://savage.nps.edu/SavageWSClientWebAppProdTest/

Subversion master source at

https://savage.nps.edu/svn/nps/Savage/services/WebServices/SAVAGEWebServicesClient/

THIS PAGE INTENTIONALLY LEFT BLANK

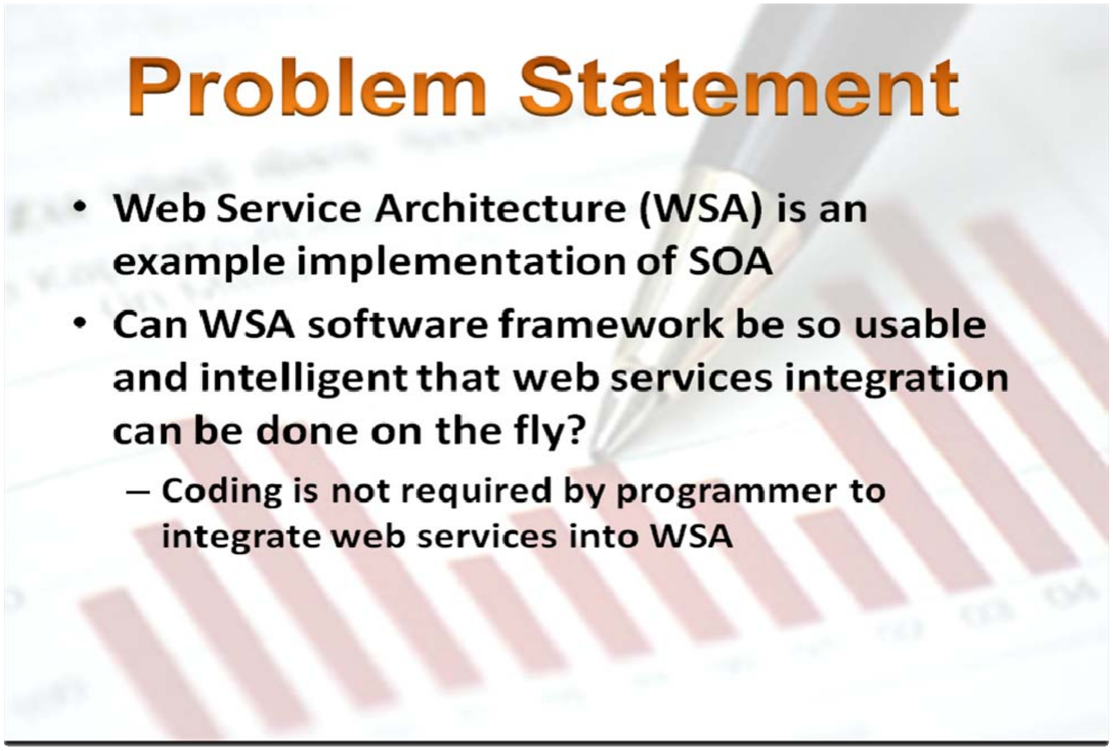
APPENDIX D. PRESENTATION SLIDESET



WEB SERVICES INTEGRATION ON THE FLY

Leong, Hoe Wai

DSO National Laboratories

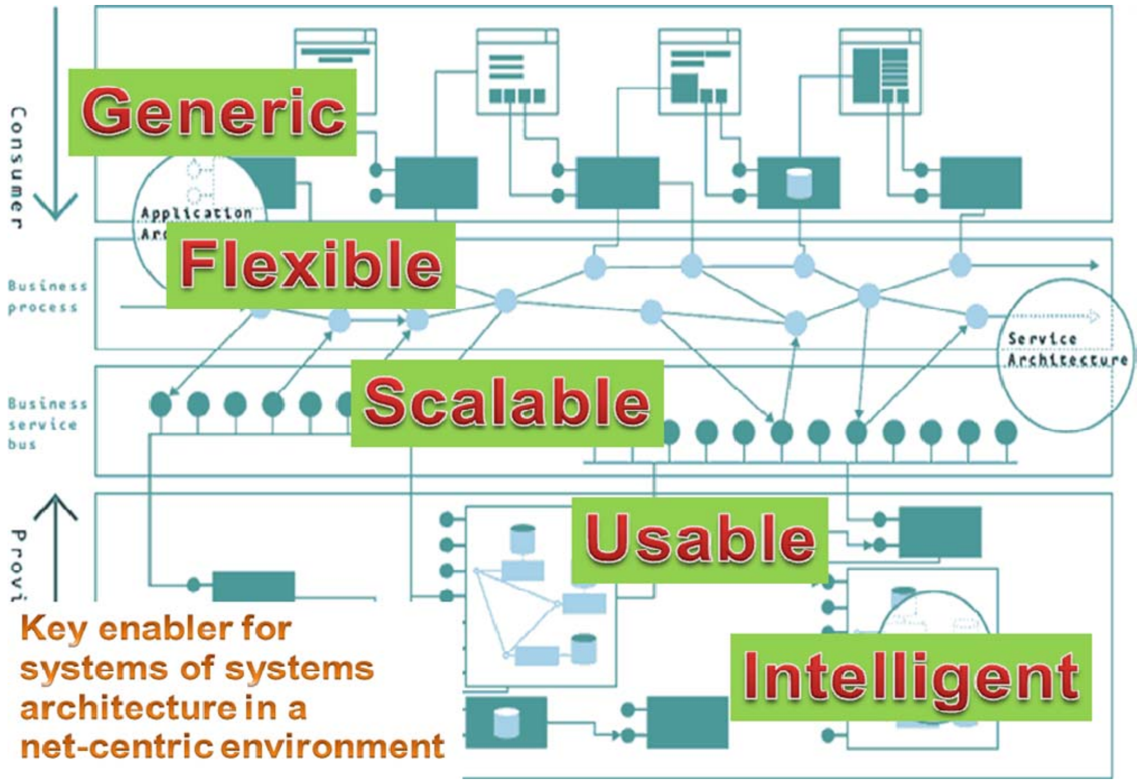


Problem Statement

- Web Service Architecture (WSA) is an example implementation of SOA
- Can WSA software framework be so usable and intelligent that web services integration can be done on the fly?
  - Coding is not required by programmer to integrate web services into WSA

Motivation

In a net centric environment, Data, tools and people operate in a distributed Network



Generic

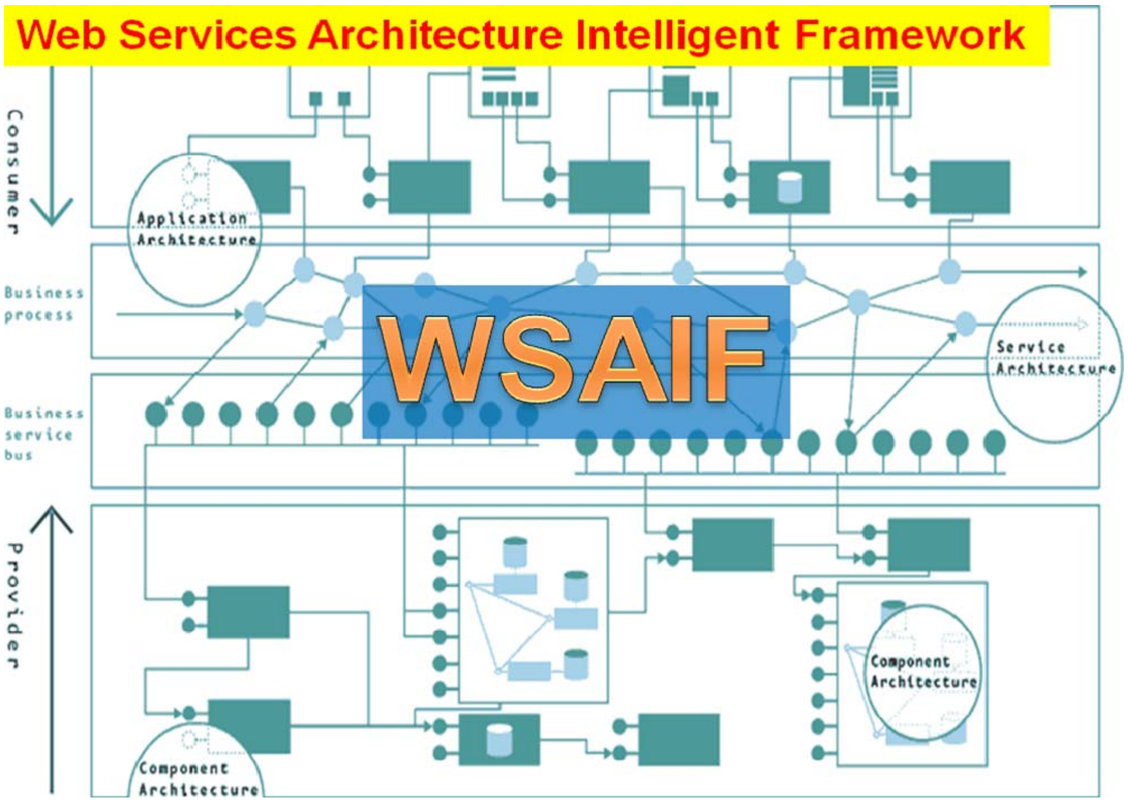
Flexible

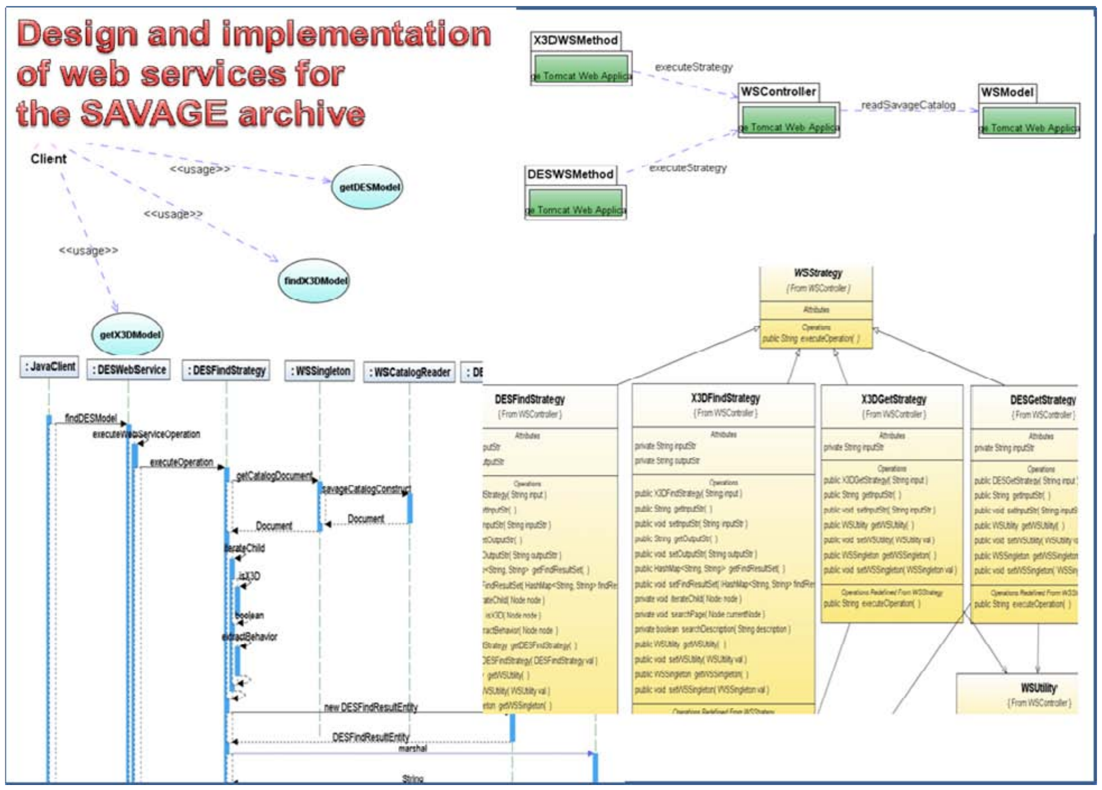
Scalable

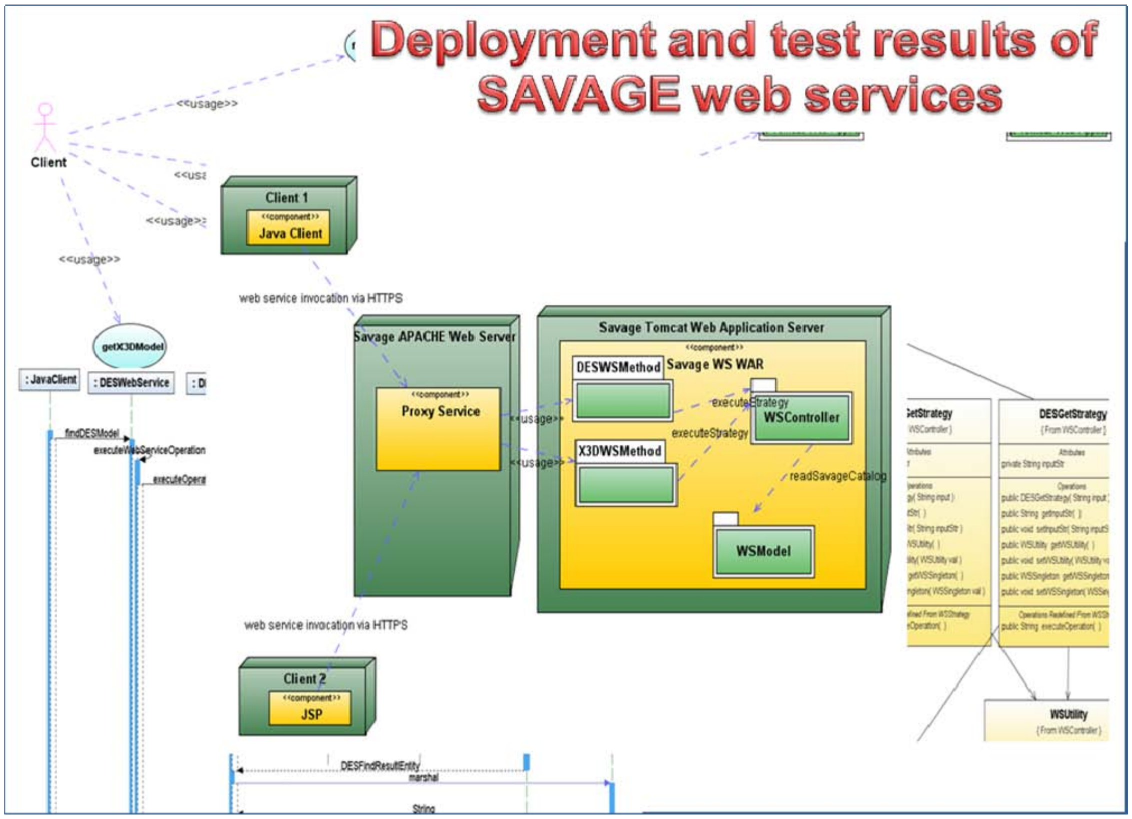
Usable

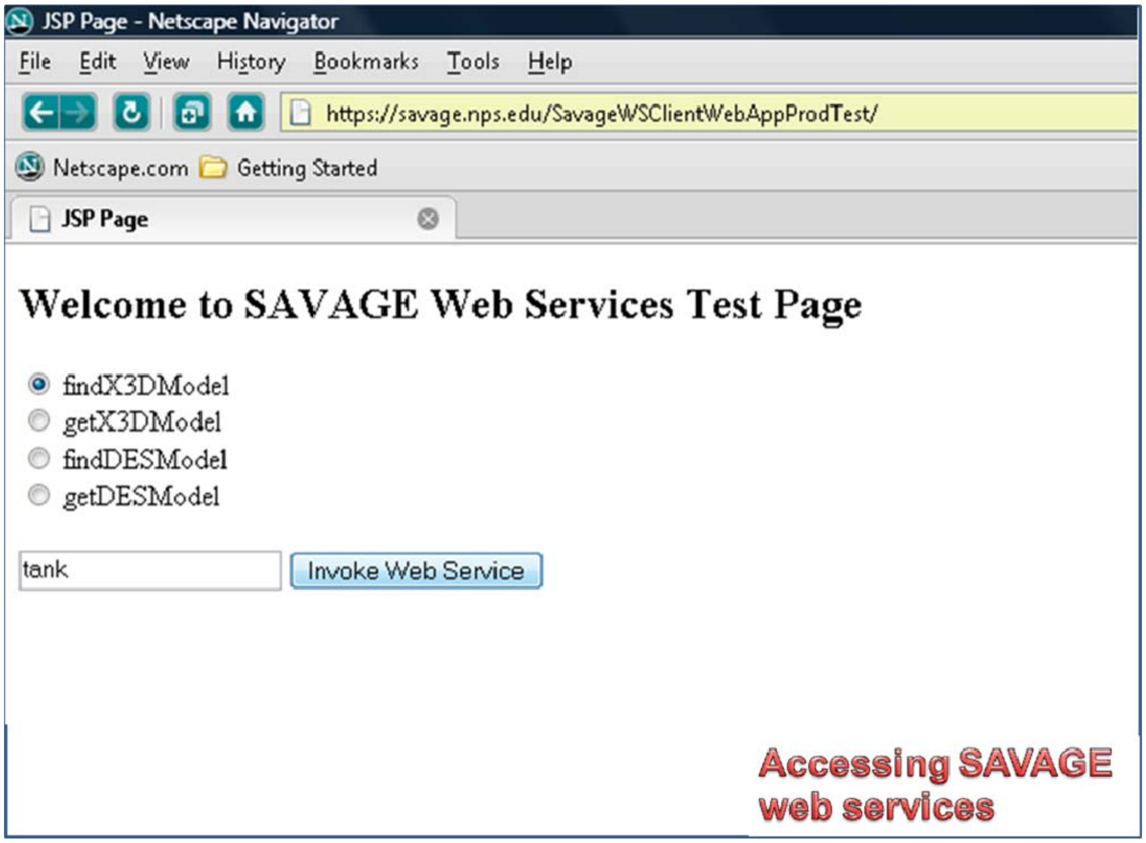
Intelligent

Key enabler for systems of systems architecture in a net-centric environment


Web Services Architecture Intelligent Framework
Consumer
Application Architecture
Business process
WSAIF
Service Architecture
Business service bus
Provider
Component Architecture
Component Architecture


Design and implementation of web services for the SAVAGE archive
Client
<<usage>>
getDESModel
<<usage>>
findX3DModel
<<usage>>
getX3DModel
X3DWSMethod
executeStrategy
WSController
readSavageCatalog
WSModel
DESWSMethod
executeStrategy
WSStrategy
: JavaClient
: DESWebService
: DESFindStrategy
: WSSingleton
: WSCatalogReader
findDESModel
executeWebServiceOperation
executeOperation
getCatalogDocument
savageCatalogConstruct
Document
Document
iterateChild
isX3D
boolean
extractBehavior
new DESFindResultEntity
DESFindResultEntity
marshal
String
DESFindStrategy
X3DFindStrategy
X3DGetStrategy
DESGetStrategy
WSUtility





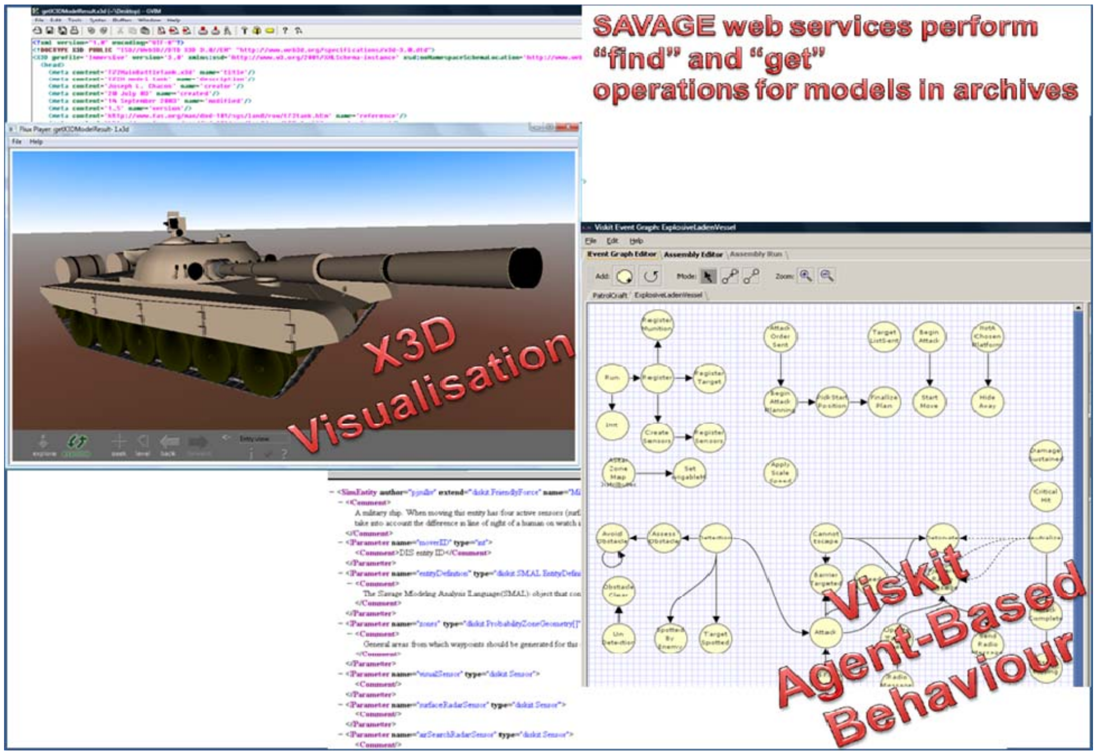
JSP Page - Netscape Navigator
File Edit View History Bookmarks Tools Help
https://savage.nps.edu/SavageWSClientWebAppProdTest/
Netscape.com Getting Started
JSP Page

Welcome to SAVAGE Web Services Test Page

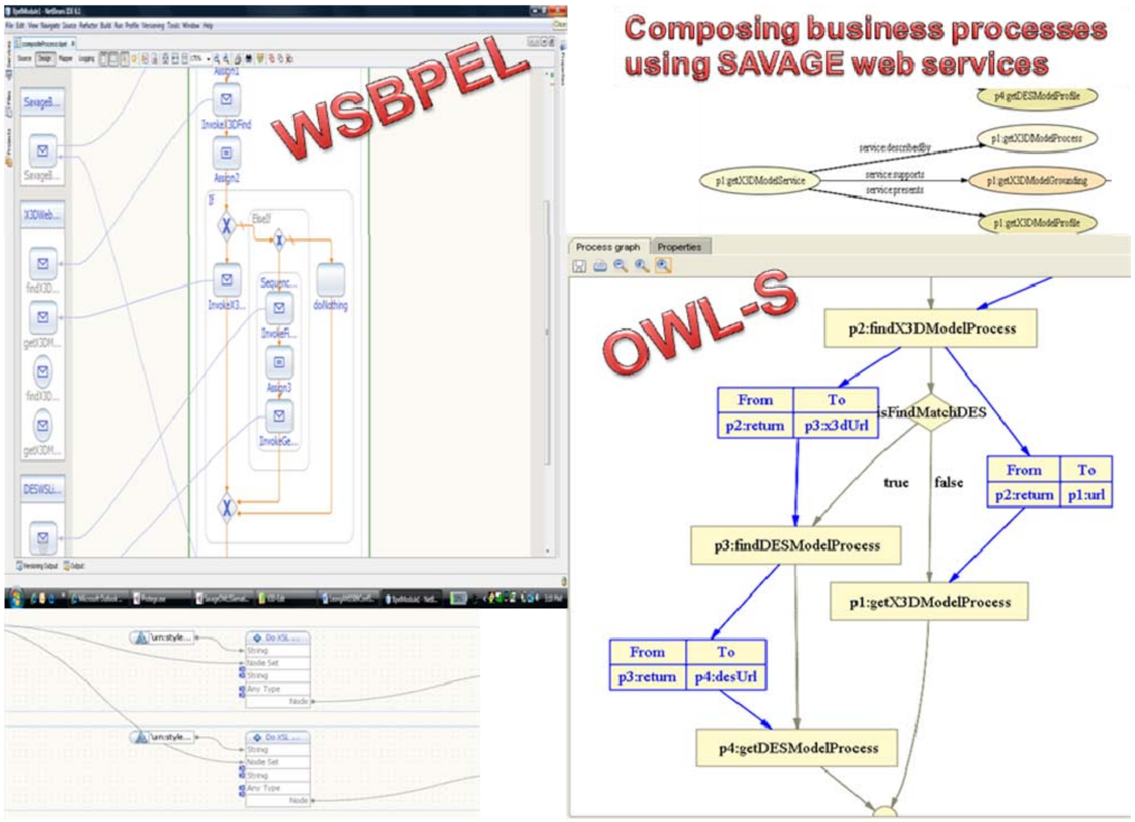
- [✓ findX3DModel]
- getX3DModel
- findDESModel
- getDESModel

tank | Invoke Web Service

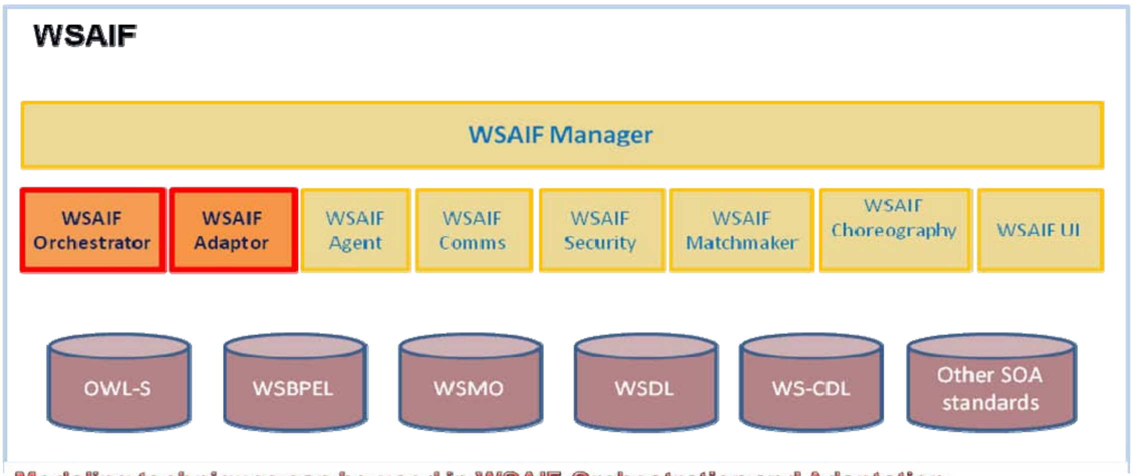
Accessing SAVAGE web services



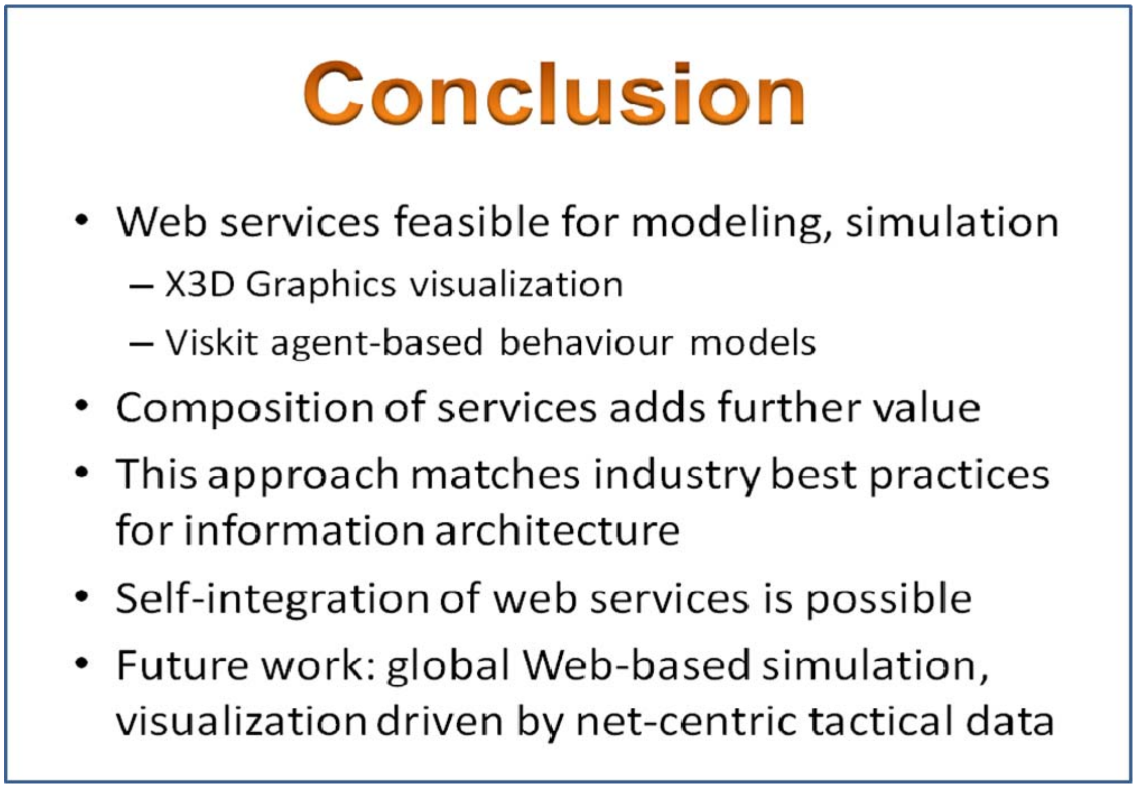
```
- <SimEntity author="yjyndiv" extend="diskit.FriendlyForce" name="Mi
  - <Comment>
      A military ship. When moving this entity has four active sensors (surf
      take into account the difference in line of sight of a human on watch i
    </Comment>
  - <Parameter name="moverID" type="int">
      <Comment>DIS entity ID</Comment>
    </Parameter>
  - <Parameter name="entityDefinition" type="diskit.SMAL.EntityDefini
    - <Comment>
        The Savage Modeling Analysis Language(SMAL) object that con
      </Comment>
    </Parameter>
  - <Parameter name="zones" type="diskit.ProbabilityZoneGeometry[]"
    - <Comment>
        General areas from which waypoints should be generated for this
      </Comment>
    </Parameter>
  - <Parameter name="visualSensor" type="diskit.Sensor">
      <Comment/>
    </Parameter>
  - <Parameter name="surfaceRadarSensor" type="diskit.Sensor">
      <Comment/>
    </Parameter>
  - <Parameter name="airSearchRadarSensor" type="diskit.Sensor">
      <Comment/>
```



The Envisioned WSAIF



Modeling techniques can be used in WSAIF Orchestration and Adaptation components.
WSAIF software agents, modeling data and supporting software infrastructure can enable web services integration on the fly



Conclusion

- Web services feasible for modeling, simulation
  - X3D Graphics visualization
  - Viskit agent-based behaviour models
- Composition of services adds further value
- This approach matches industry best practices for information architecture
- Self-integration of web services is possible
- Future work: global Web-based simulation, visualization driven by net-centric tactical data

LIST OF REFERENCES

Alesso, Smith. Developing Semantic Web Services. A K Peters Ltd, 2004.

Bell, Cesare, Lycett. Semantic Web Services Architecture for Simulation Model Reuse. IEEE 2007.

Benatallah, Casati, Grigori, Nezhad, Toumani. Developing Adapters for Web Services Integration. CAiSE 2005.

Benatallah, Nezhad. Interoperability in Semantic Web Services. SWSWPC 2004.

Berners-Lee. Weaving the Web. HarperCollins Publishers Inc., 2000.

Brickley, Guha, McBride. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004.

Brutzman and Daly. Extensible 3D Graphics For Web Authors. Morgan Kaufmann Elsevier Inc., 2007.

Burstein, Bussler, Zaremba, Finin, Huhns, Paolucci, Sheth, Williams. A Semantic Web Services Architecture, IEEE 2005.

Charfi, Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. 2004.

Childers. Applying Semantic Web Concepts to Support Net-Centric Warfare Using the Tactical Assessment Markup Language (TAML). Master's Thesis, Naval Postgraduate School, Monterey, California, 2006.

Erl. Service-Oriented Architecture (Concepts, Technology, and Design). SOA Systems Inc, 2005.

Erl. Service-Oriented Architecture (Principles of Service Design). SOA Systems Inc, 2008.

Bruijin, Bussler, Domingue, Fensel, Hepp, Kifer, Koig-Ries, Kopecky, Lara, Oren, Polleres, Scicluna, Stollberg, Roman, Lausen, Keller. Web Services Modeling Ontology (WSMO), D2 V1.2. 2005.

Foo, Wong, Ni, Leong M., Leong H. Developing a Horizon Scanning System for Early Warning. 12th ICCRTS 2007.

Gamma, Helm, Johnson, Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

Carey. XML 2nd Edition. Thomson Learning Inc., 2007.

Gorton. Essential Software Architecture. Springer-Verlag Berlin Heidelberg, 2006.

Haller, Cimpian, Mocan, Oren. WSMX – A Semantic Service-Oriented Architecture. ICWS 2005.

Hammer, Timmerman. Fundamentals of Software Integration. Jones and Bartlett Publishers Inc., 2008.

Hofmeister, Nord, Soni. Applied Software Architecture. Addison Wesley Longman Inc., 2000.

K. Harikumar, Lee, Yang, Kim, Kang. A Model for Application Integration using Web Services. IEEE 2005.

Klyne, Carroll, McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 2004.

Lara, Roman, Polleres, Fensel. A Conceptual Comparison of WSMO and OWL-S. ECOWS 2004.

Maassen. Applied Java Patterns. Sun Microsystems Inc., 2002.

Martin, Paolucci, mcIIraith, Burstein. Bringing Semantics to Web Services: The OWL-S Approach. SWSWPC 2004.

Martin, Burstein, Hobbs, Lassila, McDermott, Mcllraith, Narayanan, Paolucci, Parsia, Payne, Payne, Sirin, Srinivasan, Sycara. OWL-S: Semantic Markup for Web Services. W3C Submission, 2004.

Matthews. Bridging the SOA Divide for Deployed Assets. 2008.

Nagappan, Skoczylas, Sriganesh. Developing Java Web Services. Wiley Publishing Inc., 2003.

Paschke, Hirtle, Ginsberg, Patranjan, McCabe. 2008. “RIF Use Cases and Requirements”. W3C Working Draft, 2008.

Rao, Su. A Survey of Automated Web Service Composition Methods. SWSWPC 2004.

Rauch. Savage Modeling Analysis Language (SMAL): Metadata for Tactical Simulations and X3D Visualizations. 2006.

Sanchez, Acuna, Cavero, Marcos. Towards a UML-Compliant Semantic Web Services Development.

Schruben. Simulation Modeling with Event Graphs. ACM 1983.
-

Shafiq, Ding, Fensel. Bridging Multi Agent Systems and Web Services: towards interoperability between Software Agents and Semantic Web Services. IEEE 2006.

Skogan, Gronmo, Solheim. Web Service Composition in UML. IEEE 2004.

William. Document-Centric XML Encryption and Authorization for Coalition Messaging. NPS Thesis Proposal 2008.

Wu, Chang. Comparison of Web Services Architectures Based on Architecture Quality Properties. IEEE 2005.

Zhang, Arpinar, Aleman-Meza. Automatic Composition of Semantic Web Services.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Don Brutzman
   Naval Postgraduate School
   Monterey, California

4. Don McGregor
   Naval Postgraduate School
   Monterey, California

5. Curtis Blais
   Naval Postgraduate School
   Monterey, California

6. Christopher Priebe
   G2 Software Systems
   SPAWAR Systems Center
   San Diego, California

7. Tim Faulkuer
   TCNI
   Middletown, Maryland

8. Professor Yeo Tat Soon, Director
Temasek Defence Systems Institute
National University of Singapore
Singapore

9. Tan Lai Poh (Ms), Assistant Manager
Temasek Defence Systems Institute
National University of Singapore
Singapore